DE GRUYTER
OLDENBOURG

STUDIUM

Ethem Alpaydin

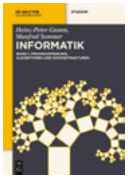
MASCHINELLES LERNEN

2. AUFLAGE

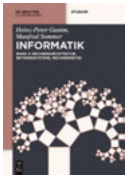
DE
G

Ethem Alpaydin
Maschinelles Lernen
De Gruyter Studium

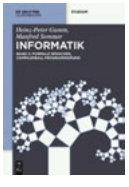
Weitere empfehlenswerte Titel



Informatik 1
H.P. Gumm, M. Sommer, 2016
ISBN 978-3-11-044227-4, e-ISBN (PDF) 978-3-11-044226-7,
e-ISBN (EPUB) 978-3-11-044231-1



Informatik 2
H.P. Gumm, M. Sommer, 2017
ISBN 978-3-11-044235-9, e-ISBN (PDF) 978-3-11-044236-6,
e-ISBN (EPUB) 978-3-11-043442-2



Informatik 3
H.P. Gumm, M. Sommer, 2019
ISBN 978-3-11-044238-0, e-ISBN (PDF) 978-3-11-044239-7,
e-ISBN (EPUB) 978-3-11-043405-7

IT-Sicherheit
C. Eckert, 2018
ISBN 978-3-11-055158-7, e-ISBN (PDF) 978-3-11-056390-0,
e-ISBN (EPUB) 978-3-11-058468-4

Künstliche Intelligenz für Ingenieure
J. Lunze, 2017
ISBN 978-3-11-044896-2, e-ISBN (PDF) 978-3-11-044897-9,
e-ISBN (EPUB) 978-3-11-044920-4

Ethem Alpaydin

Maschinelles Lernen

2. Auflage

DE GRUYTER
OLDENBOURG

Autor
Prof. Ethem Alpaydin
Department of Computer Engineering
Bogazici University
Istanbul, Turkey
alpaydin@boun.edu.tr

ISBN 978-3-11-061788-7
e-ISBN (PDF) 978-3-11-061789-4
e-ISBN (EPUB) 978-3-11-061794-8

Library of Congress Control Number: 2019939346

Bibliographic information published by the Deutsche Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Autorisierte Übersetzung der 3. Englischsprachigen Ausgabe, die bei The MIT Press, Massachusetts Institute of Technology, unter dem Titel: „Introduction to Machine Learning“ erschienen ist.

Übersetzung 1. Auflage (2008): Simone Linke
Übersetzung 2. Auflage (2018): Dr. Karen Lippert
Einbandabbildung: kishore kumar / iStock / Getty Images
Druck und Bindung: CPI books GmbH, Leck

www.degruyter.com

Inhaltsverzeichnis

Vorwort zur zweiten deutschen Auflage (dritte englische Auflage)

Das maschinelle Lernen muss zwangsläufig eines der am schnellsten wachsenden Gebiete der Computerwissenschaft sein. Nicht nur die zu verarbeitenden Datenmengen werden immer umfangreicher, sondern auch die Theorie, wie man sie verarbeiten und in Wissen verwandeln kann. In verschiedenen naturwissenschaftlichen Disziplinen, von der Astronomie bis zur Biologie, aber auch im täglichen Leben, das von digitalen Technologien immer mehr durchdrungen wird, wobei unser digitaler Fußabdruck wächst, werden ständig wachsende Mengen von Daten generiert und gesammelt. Egal, ob es sich um wissenschaftliche oder persönliche Daten handelt, ist es eine Tatsache, dass Daten, die einfach nur schlummernd herumliegen, keinerlei Nutzen haben. Kluge Köpfe haben immer neue Möglichkeiten entdeckt, wie man aus Daten Nutzen ziehen und erfolgreiche Produkte oder Dienstleistungen entwickeln kann. Bei dieser Transformationen spielt das maschinelle Lernen eine immer wichtigere Rolle.

Diese Entwicklung hat sich seit dem Erscheinen der zweiten Auflage im Jahr 2010 sogar noch verstärkt. Jedes Jahr werden die Datenmengen größer. Dabei hat nicht nur die Anzahl der Beobachtungen zugenommen, sondern es gab auch einen signifikanten Anstieg der dabei beobachteten Attribute. Die Daten sind stärker strukturiert: Es geht nicht mehr nur um Zahlen und Zeichenketten, sondern auch um Bilder, Videos, Audios, Dokumente, Webseiten, Graphen usw. Gleichzeitig entfernen sich die Daten immer weiter von den parametrischen Annahmen, von denen wir gewöhnlich ausgegangen sind – beispielsweise von der Annahme der Normalverteilung. Oftmals sind unsere Daten dynamisch, d. h., es gibt zusätzlich eine zeitliche Dimension. Manchmal erfolgen unsere Beobachtungen aus mehreren Perspektiven – für das gleiche Objekt oder Ereignis haben wir mehrere Informationsquellen, etwa in Form unterschiedlicher Sensoren oder Aufnahmeverfahren.

Es ist unsere Überzeugung, dass hinter diesen komplexen und überwältigend großen Datensammlungen einfache Erklärungen stecken. Trotz des Umfangs der Daten glauben wir, dass sie sich durch ein relativ einfaches Modell mit einer kleinen Zahl von verborgenen Faktoren und Wechselwirkungen erklären lassen. Denken wir zum Beispiel an Millionen von Kunden, die jeden Tag Tausende von Produkten kaufen, entweder online oder in ihrem lokalen Supermarkt. Die dabei anfallende Datenmenge umfasst eine riesige Anzahl von Transaktionen, doch es gibt Muster in den Daten. Menschen kaufen nicht zufallsgesteuert. Jemand, der eine Party gibt, kauft eine bestimmte Teilmenge von Produkten, und jemand, der ein Baby zu Hause hat, wird eine andere Teilmenge kaufen. Es gibt verborgene Faktoren, die das Verhalten von Kunden erklären.

Das Erschließen des verborgenen Modells aus den beobachteten Daten ist eines der Gebiete, die in den vergangenen Jahren intensiv erforscht wurden.

Die meisten Änderungen, die in der neuen Auflage vorgenommen wurden, betreffen die hier erzielten Fortschritte. Kapitel 6 enthält neue Abschnitte über Merkmalseinbettung, Singulärwertzerlegung, Faktorisierung von Matrizen, kanonische Korrelationsanalyse und Laplace-Eigenmaps.

Es gibt neue Abschnitte über Abstandsschätzungen in Kapitel 8 sowie über Kernel-Maschinen in Kapitel 13: Dimensionalitätsreduktion, Merkmalsextraktion und Abstandsschätzung sind drei Begriffe, die das gleiche Problem bezeichnen: Das ideale Abstandsmaß ist in dem Raum der idealen verborgenen Merkmale definiert, und deren Anzahl ist kleiner als die der Beobachtungswerte.

Kapitel 16 wurde umformuliert und signifikant erweitert, so dass nun auch generative Modelle mit abgedeckt sind. Wir diskutieren den Bayesschen Ansatz für alle wichtigen Modelle des maschinellen Lernens, also für die Klassifikation, die Regression, gemischte Modelle und die Dimensionalitätsreduktion. Nichtparametrische Bayessche Modelle stoßen in den letzten Jahren auf zunehmendes Interesse, weil sie es erlauben, die Komplexität des Modells an die Komplexität der Daten anzupassen.

Weitere neue Abschnitte wurden hier und da aufgenommen, meistens um neue Anwendungen oder Varianten bekannter Verfahren herauszustellen. In Kapitel 8 gibt es einen neuen Abschnitt über Ausreißererkennung. Zwei neue Abschnitte in den Kapiteln 10 und 13 widmen sich dem Problem des Rankings mithilfe von linearen Modellen bzw. Kernel-Maschinen. Nachdem in Kapitel 6 die Laplace-Eigenmaps hinzugekommen sind, habe ich mich entschlossen, auch einen neuen Abschnitt über spektrale Clusterverfahren (Kapitel 7) aufzunehmen. In Anbetracht des neu erwachten Interesses an tiefen neuronalen Netzen erschien es mir notwendig, in Kapitel 11 einen neuen Abschnitt über tiefes Lernen einzufügen. Kapitel 19 enthält einen neuen Abschnitt über multivariate Tests, mit denen verschiedene Verfahren verglichen werden können.

Seit der ersten Auflage habe ich von Lesern, die das Buch zum Selbststudium nutzen, viele Anfragen mit Bitten um Lösungen zu den Übungsaufgaben erhalten. In der vorliegenden, neuen Auflage bin ich dieser Bitte nachgekommen, indem ich zu einigen, didaktisch interessanten Übungsaufgaben Lösungen angefügt habe. Manchmal handelt es sich dabei um vollständige Lösungen, und manchmal wird nur ein Hinweis gegeben oder nur eine von mehreren möglichen Lösungen angeboten.

Ich möchte allen Dozenten und Studenten danken, die mit den beiden vorherigen Auflagen einschließlich den Übersetzungen ins Deutsche, Chinesische und Türkische sowie Reprints in Indien gearbeitet haben. Besonders bedanke ich mich bei all jenen, die mir Worte der Anerkennung, Kritik oder Errata gesendet oder mir auf anderem Wege Feedback gegeben haben. Bitte tun Sie das auch weiterhin. Meine E-Mail-Adresse ist `alpaydin@boun.edu.tr`. Die Website des Buches ist `http://www.cmpe.boun.edu.tr/~ethem/i2ml3e/`.

Es war mir auch bei der dritten Auflage ein Vergnügen, mit MIT Press zusammenzuarbeiten. Ich danke Marie Lufkin Lee, Marc Lowenthal und Kathleen Caruso für ihre Hilfe und Unterstützung.

Hinweise zur zweiten englischen Auflage

Auf dem Gebiet des maschinellen Lernens gab es seit dem Erscheinen der ersten Auflage des Buches im Jahr 2004 bedeutende Entwicklungen. Erstens sind die Anwendungsgebiete enorm gewachsen. Internetbasierte Technologien wie Suchmaschinen, Empfehlungssysteme, Spamfilter und Angriffserkennungssysteme verwenden heute routinemäßig maschinelles Lernen. In der Bioinformatik werden in immer größerem Umfang Verfahren eingesetzt, die aus vorhandenen Daten lernen. Im Bereich der natürlichen Sprachverarbeitung – beispielsweise bei der maschinellen Übersetzung – beobachten wir eine sich beschleunigende Entwicklung weg von programmierten Expertensystemen und hin zu Verfahren, die automatisch aus einem sehr großen Korpus von Beispieltexten lernen können. Auch in der Robotik, in der medizinischen Diagnostik, bei der Sprach- und Bilderkennung, in der Biometrie und im Finanzwesen gibt es immer mehr Anwendungen für die in diesem Buch vorgestellten Methoden des maschinellen Lernens, die auch unter Schlagwörter wie Mustererkennung, Data Mining und noch einigen anderen bekannt sind.

Zweitens gab es auch bei der Theorie hinter diesen Anwendungen große Fortschritte. So gestatten das Konzept der Kernel-Funktionen und die auf ihnen basierenden Kernel-Maschinen eine bessere Repräsentation des Problems. Die damit verbundene Methodik der komplexen Optimierung führt weiter als mehrlagige Perzeptronen mit sigmoidalen verborgenen Einheiten, die mittels Gradientenabstieg trainiert werden. Bayessche Methoden mit geeignet gewählten a-priori-Verteilungen fügen Expertenwissen zu dem hinzu, was uns die Daten verraten. Graphenmodelle erlauben die Darstellung durch ein Netzwerk von Knoten, und mit effizienten Inferenzalgorithmen sind Suchanfragen an solche Netzwerke möglich. Diese Entwicklungen erfordern es, die drei Themen – Kernel-Maschinen, Bayessche Schätzung und Graphenmodelle –, die in der ersten Auflage in Abschnitten behandelt wurden, ausführlicher zu betrachten, weshalb drei neue Kapitel hinzugekommen sind.

Von großer Bedeutung für das maschinelle Lernen war die Erkenntnis, dass Experimente auf der Basis maschinellen Lernens ein besseres Design brauchen. Aus diesem Grund habe ich in der zweiten Auflage das Kapitel über statistische Tests so umformuliert, dass es nun auch das Design und die Analyse von Experimenten mit maschinellem Lernen umfasst. Wichtig ist, dass das Testen nicht als separater Schritt aufgefasst werden sollte, der ausgeführt wird, wenn alle Durchläufe abgeschlossen sind (auch wenn das neue Kapitel am Ende des Buches steht); vielmehr sollte man den gesamten Prozess des Experimentierens vorab konzipieren, relevante

Es war mir auch bei der dritten Auflage ein Vergnügen, mit MIT Press zusammenzuarbeiten. Ich danke Marie Lufkin Lee, Marc Lowenthal und Kathleen Caruso für ihre Hilfe und Unterstützung.

Hinweise zur zweiten englischen Auflage

Auf dem Gebiet des maschinellen Lernens gab es seit dem Erscheinen der ersten Auflage des Buches im Jahr 2004 bedeutende Entwicklungen. Erstens sind die Anwendungsgebiete enorm gewachsen. Internetbasierte Technologien wie Suchmaschinen, Empfehlungssysteme, Spamfilter und Angriffserkennungssysteme verwenden heute routinemäßig maschinelles Lernen. In der Bioinformatik werden in immer größerem Umfang Verfahren eingesetzt, die aus vorhandenen Daten lernen. Im Bereich der natürlichen Sprachverarbeitung – beispielsweise bei der maschinellen Übersetzung – beobachten wir eine sich beschleunigende Entwicklung weg von programmierten Expertensystemen und hin zu Verfahren, die automatisch aus einem sehr großen Korpus von Beispieltexten lernen können. Auch in der Robotik, in der medizinischen Diagnostik, bei der Sprach- und Bilderkennung, in der Biometrie und im Finanzwesen gibt es immer mehr Anwendungen für die in diesem Buch vorgestellten Methoden des maschinellen Lernens, die auch unter Schlagwörter wie Mustererkennung, Data Mining und noch einigen anderen bekannt sind.

Zweitens gab es auch bei der Theorie hinter diesen Anwendungen große Fortschritte. So gestatten das Konzept der Kernel-Funktionen und die auf ihnen basierenden Kernel-Maschinen eine bessere Repräsentation des Problems. Die damit verbundene Methodik der komplexen Optimierung führt weiter als mehrlagige Perzeptronen mit sigmoidalen verborgenen Einheiten, die mittels Gradientenabstieg trainiert werden. Bayessche Methoden mit geeignet gewählten a-priori-Verteilungen fügen Expertenwissen zu dem hinzu, was uns die Daten verraten. Graphenmodelle erlauben die Darstellung durch ein Netzwerk von Knoten, und mit effizienten Inferenzalgorithmen sind Suchanfragen an solche Netzwerke möglich. Diese Entwicklungen erfordern es, die drei Themen – Kernel-Maschinen, Bayessche Schätzung und Graphenmodelle –, die in der ersten Auflage in Abschnitten behandelt wurden, ausführlicher zu betrachten, weshalb drei neue Kapitel hinzugekommen sind.

Von großer Bedeutung für das maschinelle Lernen war die Erkenntnis, dass Experimente auf der Basis maschinellen Lernens ein besseres Design brauchen. Aus diesem Grund habe ich in der zweiten Auflage das Kapitel über statistische Tests so umformuliert, dass es nun auch das Design und die Analyse von Experimenten mit maschinellem Lernen umfasst. Wichtig ist, dass das Testen nicht als separater Schritt aufgefasst werden sollte, der ausgeführt wird, wenn alle Durchläufe abgeschlossen sind (auch wenn das neue Kapitel am Ende des Buches steht); vielmehr sollte man den gesamten Prozess des Experimentierens vorab konzipieren, relevante

Faktoren definieren, den richtigen Versuchsablauf wählen und erst dann die Durchläufe ausführen und die Daten analysieren.

Lange Zeit war die vorherrschende Meinung in der wissenschaftlichen Community, dass unser gegenwärtiges Wissen im Allgemeinen und das Wissen in der Computerwissenschaft im Besonderen nicht ausreicht, um Maschinen zu entwickeln, die so intelligent sind wie wir und dass künstliche Intelligenz aus diesem Grund nicht Realität werden könnte. Die meisten Leute glauben, dass wir eine neue Technologie brauchen, ein neues Material, neuartige Berechnungsmechanismen oder neuartige Programmiermethoden. Demnach könnten wir bis dahin nur einige Aspekte der menschlichen Intelligenz „simulieren“ und auch dies nur in eingeschränkter Form, weshalb sie letzlich unerreichbar bliebe.

Ich glaube, dass sich diese Ansicht schon bald als falsch erweisen wird. Zuerst haben wir dies beim Schach gesehen, und inzwischen beobachten wir Ähnliches in vielen Gebieten. Wenn wir nur genügend Speicher und Rechenleistung haben, können wir Aufgaben mithilfe relativ einfacher Algorithmen realisieren. Der Trick dabei ist das Lernen, entweder aus Beispieldaten oder durch bestärkendes Lernen nach dem Prinzip von Versuch und Irrtum. Es sieht danach aus, dass der Einsatz von Algorithmen für überwachtes und nahezu unüberwachtes Lernen – etwa im Bereich der maschinellen Übersetzung – schon bald möglich sein wird. Das Gleiche gilt für viele andere Gebiete, beispielsweise für die unbemannte Navigation in der Robotik mittels bestärkendem Lernen. Ich glaube, dass sich diese Entwicklung in vielen Bereichen der künstlichen Intelligenz fortsetzen wird, und der Schlüssel ist das Lernen. Wir müssen keine neuen Algorithmen einführen, wenn Maschinen selbst in der Lage sind zu lernen; wir müssen sie nur mit ausreichend großen Datenmengen füttern und genügend Rechenleistung zur Verfügung haben.

Ich möchte an dieser Stelle allen Lehrenden und Studierenden danken, die mit der ersten Auflage einschließlich dem Reprint in Indien und der deutschen Übersetzung gearbeitet haben. Besonders danke ich jenen, die mir Worte der Anerkennung und Errata zukommen ließen oder mir in anderer Weise Feedback gegeben haben. Bitte tun Sie das auch weiterhin. Meine E-Mail-Adresse ist `alpaydin@boun.edu.tr`.

Die zweite Auflage bietet auch im Internet in größerem Umfang Unterstützung unter `http://www.cmpe.boun.edu.tr/~ethem/i2ml`.

Danken möchte ich außerdem meinen ehemaligen und aktuellen Doktoranden Mehmet Gönen, Esma Kılıç, Murat Semerci, M. Aydın Ulaş und Olcay Taner Yıldız sowie den Hörern meiner Vorlesungen. Wenn man sein Wissen in einem bestimmten Fach testen möchte, ist der beste Weg der, dieses Fach zu unterrichten.

Es war mir auch bei der zweiten Auflage ein Vergnügen, mit MIT Press zu arbeiten. Ich danke Bob Prior, Ada Brunstein, Erin K. Shoudy, Kathleen Caruso und Marcy Ross für ihre Hilfe und Unterstützung.

Vorwort zur ersten Auflage

Maschinelles Lernen bedeutet, Computer so zu programmieren, dass ein Leistungskriterium auf der Basis eines Beispiels oder früherer Erfahrungswerte optimiert wird. Lernen ist dann nötig, wenn wir für das Lösen eines bestimmten Problems nicht unmittelbar ein Computerprogramm schreiben können, sondern zunächst Beispieldaten oder Erfahrungswerte betrachten müssen. Eine solche Situation liegt etwa dann vor, wenn keine menschliche Expertise vorhanden ist oder wenn Menschen nicht in der Lage sind, ihre Expertise zu erklären. Betrachten wir zum Beispiel die natürliche Spracherkennung, d. h. die Umwandlung akustischer Sprachsignale in ASCII-Text. Diese Aufgabe bereitet uns offensichtlich keine Schwierigkeiten, doch wir können nicht erklären, wie wir sie lösen. Verschiedene Personen sprechen ein und dasselbe Wort unterschiedlich aus, wobei Alter, Geschlecht oder Dialekt eine Rolle spielen. Der Ansatz des maschinellen Lernens besteht darin, eine große Sammlung von Sprachäußerungen anzulegen und dann zu lernen, diesen Sprachäußerungen Wörter zuzuordnen.

Ein weiterer Fall ist der, dass das zu lösende Problem sich zeitlich ändert oder von bestimmten Umgebungsbedingungen abhängig ist. Für solche Probleme hätten wir gern universelle Systeme, die sich an die jeweiligen Umstände anpassen können. Denken wir etwa an Datenpakete, die in einem Computernetzwerk übertragen werden. Der Pfad, welcher die Servicequalität von der Quelle zum Ziel maximiert, wird sich in Abhängigkeit von der Netzwerkauslastung fortwährend ändern. Ein Lernprogramm für das Routing ist in der Lage, sich an den besten Pfad anzupassen, indem es die Netzwerkauslastung überwacht. Ein anderes Beispiel ist eine intelligente Benutzeroberfläche, die sich an die biometrischen Daten der Benutzer anpassen kann, d. h. an Merkmale wie Dialekt, Handschrift oder Arbeitsweise.

Bereits heute gibt es in vielen Bereichen nützliche Anwendungen des maschinellen Lernens. Dazu zählen kommerziell verfügbare Sprach- und Handschrifterkennungssysteme. Einzelhandelsunternehmen analysieren ihre Verkaufsdaten, um daraus etwas über das Verhalten ihrer Kunden zu lernen und so die Kundenbeziehung zu verbessern. Finanzinstitute analysieren Transaktionen, um die Kreditausfallrisiken ihrer Kunden einschätzen zu können. Roboter lernen, ihr eigenes Verhalten zu optimieren, um die ihnen gestellten Aufgaben mit minimalem Ressourcenaufwand zu erledigen. In der Bioinformatik ist es nur mit Computern möglich, die vorhandenen, riesigen Datenmengen zu analysieren und Wissen daraus zu extrahieren. Dies sind nur einige der Anwendungsfelder, auf die in dem vorliegenden Buch immer wieder Bezug benommen wird. Wir können uns zumindest vorstellen, welche zukünftigen Anwendungen durch maschinelles Lernens realisiert werden: autonome Fahrzeuge, die sich auf unterschiedliche Straßen- und Wetterverhältnisse einstellen, Telefone, die Übersetzungen aus einer bzw. in eine Fremdsprache in Echtzeit ausfüh-

ren, autonome Roboter, die in einer unbekannten Umgebung navigieren, zum Beispiel auf der Oberfläche eines anderen Planeten. Das maschinelle Lernen wird mit Sicherheit ein spannendes Arbeitsgebiet bleiben!

In dem vorliegenden Buch werden viele Methoden vorgestellt, deren Grundlagen aus unterschiedlichen Forschungsgebieten stammen: Statistik, Mustererkennung, neuronale Netze, künstliche Intelligenz, Signalverarbeitung, Regelungstechnik und Data Mining. In der Vergangenheit folgten die Forscher der genannten Gebiete unterschiedlichen Pfaden und setzten unterschiedliche Schwerpunkte. Dieses Buch hat das Ziel, die verschiedenen Ansätze zu verbinden und eine vereinheitlichte Behandlung der Probleme sowie Lösungsvorschläge vorzulegen.

Es handelt sich hierbei um ein einführendes Lehrbuch, das für Kurse über maschinelles Lernen im fortgeschrittenen Bachelorstudium oder am Anfang des Masterstudiums gedacht ist. Es wendet sich auch an Ingenieure, die in der Industrie arbeiten und sich für die Anwendung der hier diskutierten Verfahren interessieren. Vorausgesetzt werden Vorkenntnisse über Programmierung, Wahrscheinlichkeitsrechnung, Analysis und lineare Algebra. Es wurde angestrebt, alle behandelten Lernalgorithmen so ausführlich zu erklären, dass es nur noch ein kleiner Schritt ist, die im Buch angegebenen Gleichungen in ein Computerprogramm zu überführen. In einigen Fällen wurden zusätzlich Pseudocodes beigefügt, wodurch diese Aufgabe noch einfacher wird.

Das Buch kann als Grundlage für eine einsemestrige Vorlesung dienen, indem ausgewählte Themen aus den verschiedenen Kapiteln behandelt werden. Auch für eine zweisemestrige Vorlesung bietet das Buch ausreichend Stoff, der durch zusätzliche Forschungsaufsätze vertieft werden kann; in diesem Fall hoffe ich, dass die Referenzen am Ende jedes Kapitels hilfreich sind.

Unter `http://www.cmpe.boun.edu.tr/~ethem/i2ml/` finden Sie die Website des Buches, auf der ich für das Thema relevante Informationen veröffentlichen werde, die mir nach Drucklegung des Buches bekannt geworden sind, beispielsweise Errata. Ihr Feedback ist mir unter meiner E-Mail-Adresse alpaydin@boun.edu.tr willkommen.

Es hat mir viel Freude bereitet, dieses Buch zu schreiben, und ich hoffe, dass es Ihnen beim Lesen ebenso ergeht.

Notationen

x	Skalarwert
$\boldsymbol{x}$	Vektor
$\mathbf{X}$	Matrix
$\boldsymbol{x}^T$	Transponierte
$\mathbf{X}^{-1}$	Inversion
X	Zufallsvariable
$P(X)$	Wahrscheinlichkeitsfunktion, wenn X diskret ist
$p(X)$	Wahrscheinlichkeitsdichtefunktion, wenn X kontinuierlich ist
$P(X\|Y)$	Bedingte Wahrscheinlichkeit von X gegeben Y
$E[X]$	Erwartungswert der Zufallsvariable X
$\text{Var}(X)$	Varianz von X
$\text{Cov}(X,Y)$	Kovarianz von X und Y
$\text{Corr}(X,Y)$	Korrelation von X und Y
μ	Mittelwert
σ^2	Varianz
$\boldsymbol{\Sigma}$	Kovarianzmatrix
m	Mittelwertschätzer
s^2	Varianzschätzer
$\mathbf{S}$	Schätzer für die Kovarianzmatrix
$\mathcal{N}(\mu,\sigma^2)$	Univariate Normalverteilung mit Mittelwert μ und Varianz σ^2
$\mathcal{Z}$	Einheitsnormalverteilung: $\mathcal{N}(0,1)$
$\mathcal{N}_d(\boldsymbol{\mu},\boldsymbol{\Sigma})$	d-variate Normalverteilung mit Mittelwertvektor $\boldsymbol{\mu}$ und Kovarianzmatrix $\boldsymbol{\Sigma}$
x	Eingabe
d	Anzahl der Eingaben: Eingabedimensionalität
y	Ausgabe
r	gewünschte Ausgabe
K	Anzahl der Ausgaben (Klassen)
N	Anzahl der Trainingsinstanzen
z	Verborgener Wert, intrinsische Dimension, latenter Faktor
k	Anzahl der verborgenen Dimensionen, latenten Faktoren
$\mathcal{C}_i$	Klasse i

$\mathcal{X}$	Trainingsstichprobe
$\{x^t\}_{t=1}^N$	Menge von Eingaben x indiziert mit t im Intervall von 1 bis N
$\{x^t, r^t\}_t$	Menge an geordneten Paaren von Eingabe und gewünschter Ausgabe mit Index t
$g(x\|\theta)$	Funktion von x definiert durch die Parametermenge θ
$\underset{\theta}{\operatorname{argmax}}\, g(x\|\theta)$	Das Argument θ, für das g den Maximalwert annimmt
$\underset{\theta}{\operatorname{argmin}}\, g(x\|\theta)$	Das Argument θ, für das g den Minimalwert annimmt
$E(\theta\|\mathcal{X})$	Fehlerfunktion mit Parametern θ an der Stichprobe $\mathcal{X}$
$l(\theta\|\mathcal{X})$	Likelihood der Stichprobe $\mathcal{X}$ gegeben die Parameter θ
$\mathcal{L}(\theta\|\mathcal{X})$	Log-Likelihood der Stichprobe $\mathcal{X}$ gegeben die Parameter θ
$1(c)$	1, falls c wahr ist, andernfalls 0
$\#\{c\}$	Anzahl an Elementen, für die c wahr ist
δ_{ij}	Kronecker-Delta: 1, falls $i = j$, andernfalls 0

1 Einführung

1.1 Was ist maschinelles Lernen?

Wir leben im Zeitalter von Big Data. Früher waren Daten eine Angelegenheit für Unternehmen, und es gab Rechenzentren, in denen Daten gespeichert und verarbeitet wurden. Mit dem Siegeszug des Personalcomputers und später mit der allgemeinen Verfügbarkeit drahtloser Kommunikationstechnologien wurden wir alle mehr und mehr zu Produzenten von Daten. Jedes Mal, wenn wir ein Produkt kaufen, einen Film ausleihen, einen Blogeintrag oder einen Post in einem sozialen Netzwerk schreiben, ja sogar, wenn wir einfach nur herumlaufen oder -fahren, dann generieren wir Daten.

Dabei ist jeder von uns nicht nur Produzent, sondern gleichzeitig Verbraucher von Daten. Wir wollen Produkte und Dienstleistungen haben, die auf uns zugeschnitten sind. Wir wollen, dass unsere Bedürfnisse verstanden und unsere Wünsche vorhergesagt werden. Denken wir zum Beispiel an eine Supermarktkette, die Tausende von Waren an Millionen von Kunden verkauft – entweder in ein paar Hundert herkömmlichen Filialen oder über einen virtuellen Laden im Netz. Die Details jeder Transaktion werden gespeichert: das Datum, die Kundennummer, die gekauften Produkte und ihre jeweilige Menge, der insgesamt gezahlte Betrag usw. Die Supermarktkette ist daran interessiert vorherzusagen, welcher Kunde wahrscheinlich welches Produkt kaufen wird, und auf diese Weise ihren Umsatz und ihren Gewinn zu maximieren. Die Kunden wiederum sind daran interessiert, diejenigen Produkte zu finden, die ihren Bedürfnissen am besten entsprechen.

Es ist nicht offensichtlich, wie diese Anforderungen erfüllt werden können. Wir wissen nicht genau, welche Kunden eine bestimmte Sorte Eiscreme oder das nächste Buch eines bestimmten Autors kaufen werden. Wir wissen nicht, ob sie diesen neuen Film ansehen oder jene Stadt besuchen werden; ebenso wenig wissen wir, welche Links sie anklicken werden. Das Verhalten von Kunden variiert zeitlich und räumlich. Wir wissen allerdings, dass es nicht völlig zufällig ist. Die Menschen gehen nicht in den Supermarkt, um dort zufällig irgendwelche Dinge zu kaufen. Wenn sie Bier kaufen, dann kaufen sie auch Chips; sie kaufen Eis im Sommer und Gewürze für Glühwein im Winter. Es gibt also gewisse Muster in den Daten.

Um ein Problem auf einem Computer zu lösen, benötigen wir einen Algorithmus. Ein Algorithmus ist eine Folge von Anweisungen, die ausgeführt werden muss, um eine Eingabe in eine Ausgabe zu transformieren. Beispielsweise kann man einen Algorithmus entwerfen, der das Problem des Sortierens löst. Die Eingabe ist eine Menge von Zahlen und die Ausgabe ist eine geordnete Liste dieser Zahlen. Für eine bestimmte Aufgabe kann es eine Vielzahl von Algorithmen geben. Dann sind wir interessiert daran, den effizientesten dieser Algorithmen zu finden, also denjenigen, der die wenigsten Anweisungen oder den geringsten Speicherplatz oder beides benötigt.

Es gibt jedoch Aufgaben, für die wir keinen Algorithmus haben. Ein Beispiel hierfür ist die Vorhersage des Kundenverhaltens, ein anderes die Unterscheidung zwischen Spam-Mails und legitimen E-Mails. Wir wissen, dass die Eingabe im zweiten Beispiel aus einer E-Mail besteht, die wir im einfachsten Fall als eine aus Zeichen bestehende Datei auffassen können. Wir wissen auch, was wir als Ausgabe erwarten: eine ja/nein-Aussage, die angibt, ob es sich bei der Nachricht um Spam handelt oder nicht. Wir wissen jedoch nicht, wie wir die Eingabe in die Ausgabe transformieren können. Denn was als Spam betrachtet wird, ändert sich mit der Zeit und unterscheidet sich von Individuum zu Individuum.

Was uns an Wissen fehlt, ersetzen wir durch Daten. Es ist einfach, Tausende von Beispiel-Mails zusammenzutragen, wobei wir von einigen dieser Mails wissen, dass sie Spam sind, und von anderen, dass es sich um legitime Mails handelt. Was wir wollen, ist zu „lernen“, was Spam ausmacht. Mit anderen Worten, wir möchten, dass der Computer (die Maschine) automatisch den Algorithmus für diese Aufgabe herausfindet. Es besteht kein Bedarf, das Sortieren von Zahlen zu lernen, da es für diese Aufgabe bereits Algorithmen gibt. Doch es gibt viele Anwendungen, für die wir keinen Algorithmus, aber eine Menge von Daten haben.

Wir mögen vielleicht nicht in der Lage sein, den Prozess komplett zu identifizieren, aber wir glauben, dass wir eine *gute und nützliche Näherung* konstruieren können. Diese Approximation vermag nicht, alles zu erklären, aber könnte dennoch in der Lage sein, für einen Teil der Daten eine Erklärung zu liefern. Obwohl es zwar vielleicht unmöglich ist, den kompletten Prozess zu identifizieren, glauben wir dennoch, dass wir gewisse Muster oder Regelmäßigkeiten aufspüren können. Genau darin besteht die Marktlücke für das maschinelle Lernen. Solche Muster können uns helfen, den Prozess zu verstehen, oder wir können sie nutzen, um Vorhersagen zu treffen. Unter der Annahme, dass die Zukunft, zumindest die nahe Zukunft, sich nicht großartig von dem vergangenen Zeitpunkt unterscheidet, an dem die Daten gesammelt wurden, kann von Prognosen erwartet werden, dass sie ebenfalls richtig sind.

Die Anwendung von Methoden des maschinellen Lernens auf größere Datenbanken nennt man *Data Mining*. Die Analogie liegt darin, dass eine große Menge von Rohmaterial aus einer Mine extrahiert wird, die dann

durch Verarbeitung zu einer kleinen Menge sehr wertvollen Materials wird. Ähnlich dazu wird beim Data Mining ein großes Datenvolumen verarbeitet, um ein einfaches Modell von hohem Nutzen, beispielsweise mit einer hohen Vorhersagegenauigkeit, zu gewinnen. Die Anwendungsgebiete sind zahllos. Neben dem Warenhandel analysieren u.a. Vermögensinstitute ihre erhobenen Daten, um Modelle zu konstruieren, die sie bei Kreditanwendungen, beim Aufspüren von Schwindel und an der Börse einsetzen können. In Manufakturen werden Lernmodelle zur Optimierung, Steuerung und Fehlerbehebung benutzt. In der Medizin erweisen sich Lernprogramme bei der medizinischen Diagnose als nützlich. Im Telekommunikationsbereich werden Anrufmuster untersucht, um Netzwerke zu optimieren und die Dienstgüte zu maximieren. In der Wissenschaft können große Datenmengen der Physik, Astronomie und Biologie nur mittels Computern schnell genug analysiert werden. Das World Wide Web ist riesig; es wächst ständig weiter und die Suche nach relevanter Information kann nicht mehr manuell vorgenommen werden.

Aber das maschinelle Lernen ist nicht nur für den Datenbankbereich relevant, es fällt ebenfalls in das Gebiet der künstlichen Intelligenz. Um von Intelligenz sprechen zu können, muss ein System in einer sich verändernden Umwelt in der Lage sein, zu lernen. Wenn das System lernen und sich an derlei Veränderungen anpassen kann, muss der Systementwickler nicht jede erdenkliche Situation vorhersehen und eine jeweils angemessene Lösung bereitstellen.

Maschinelles Lernen hilft uns auch, Lösungen für eine Vielzahl von Problemen im visuellen Bereich, der Spracherkennung, und der Robotik zu finden. Nehmen wir einmal die Gesichtserkennung als Beispiel: dies ist eine Aufgabe, die wir selbst mühelos bewältigen; jeden Tag erkennen wir Familienmitglieder und Freunde anhand ihrer Gesichter oder anhand von Fotos, und das trotz Unterschieden in Haltung, Beleuchtung, Frisur, und so weiter. Aber wir tun dies unbewusst und sind nicht in der Lage, zu erklären, wie wir es tun. Und weil wir eben nicht fähig sind, unsere Expertise in Worte zu fassen, können wir kein entsprechendes Computerprogramm schreiben. Gleichzeitig wissen wir jedoch, dass ein Gesicht nicht einfach eine zufällige Ansammlung von Pixeln ist; ein Gesicht hat eine gewisse Struktur. Es ist symmetrisch. Da sind die Augen, die Nase, der Mund – alle jeweils an bestimmten Stellen im Gesicht. Das Gesicht eines jeden Menschen ist ein Muster, welches sich aus einer ganz bestimmten Kombination all dieser Komponenten zusammensetzt. Indem Beispielbilder vom Gesicht einer Person analysiert werden, erfasst ein Lernprogramm das für die Person spezifische Muster und erkennt die Person dann dadurch, dass ein gegebenes Bild nach genau diesem Muster durchsucht wird. Dies ist ein Beispiel aus dem Gebiet der *Mustererkennung*.

Maschinelles Lernen heißt, Computer so zu programmieren, dass ein bestimmtes Leistungskriterium anhand von Beispieldaten oder Erfahrungswerten aus der Vergangenheit optimiert wird. Wir haben ein Modell

mit bestimmten Parametern definiert, und Lernen heißt in diesem Fall, ein Computerprogramm auszuführen, um die Parameter des Modells unter Verwendung von Trainingsdaten oder Erfahrungswerten der Vergangenheit zu optimieren. Das Modell mag *prädiktiv* sein, um zukünftige Vorhersagen zu treffen, oder *deskriptiv*, um Wissen aufgrund der Daten zu erlangen, oder beides.

Maschinelles Lernen nutzt die Theorien der Statistik bei der Konstruktion von mathematischen Modellen, da die Kernaufgabe darin besteht, aufgrund einer Stichprobe Schlussfolgerungen zu ziehen. Die Rolle der Informatik ist dabei zweigeteilt: zum einen benötigen wir für das Training effiziente Algorithmen, um das Optimierungsproblem zu lösen und um die massiven Datenmengen, mit denen wir zumeist arbeiten, zu speichern und zu verarbeiten. Zum anderen müssen nach dem Erlernen eines Modells seine Darstellung und algorithmische Lösung für das Herleiten von Rückschlüssen ebenfalls so effizient wie möglich sein. Bei bestimmten Anwendungen kann die Effizienz des Lern- oder Inferenzalgorithmus, sprich seine räumliche und zeitliche Komplexität, genauso wichtig sein wie seine Vorhersagegenauigkeit.

Widmen wir uns nun einigen Beispielanwendungen im Detail, um ein tieferes Verständnis für die Arten und Verwendungsmöglichkeiten des maschinellen Lernens zu gewinnen.

1.2 Beispiele für Anwendungen des maschinellen Lernens

1.2.1 Erlernen von Assoziationen

Im Falle des Warenhandels – beispielsweise bei einer Supermarktkette – besteht eine Anwendung des maschinellen Lernens in der *Warenkorbanalyse*, bei der versucht wird, Assoziationen zwischen Produkten, die ein Kunde gekauft hat, zu finden: Wenn Kunden, die X kaufen, typischerweise auch Y kaufen und es einen Kunden gibt, der X, aber nicht Y kauft, dann kann dieser Kunde als potenzieller Käufer für Y erachtet werden. Haben wir einmal solche Kunden gefunden, dann können wir sie gezielt in Cross-Selling-Versuche einbeziehen.

Assoziationsregel

Das Finden einer *Assoziationsregel* entspricht unserem Interesse, eine bedingte Wahrscheinlichkeit der Form $P(Y|X)$ zu lernen, wobei Y das Produkt ist, das durch X bedingt sein soll. X steht für die Ware oder Warengruppe, von der wir schon wissen, dass der Kunde sie gekauft hat.

Nehmen wir nun einmal an, dass wir anhand unserer Daten auf die Gleichung $P(\text{Chips}|\text{Bier}) = 0{,}7$ kommen. Dann können wir die folgende Regel definieren:

> 70 Prozent aller Kunden, die Bier kaufen, kaufen auch Chips.

Möglicherweise wollen wir nun noch genauer zwischen Kunden unterscheiden, und um dies zu erreichen, berechnen wir $P(Y|X, D)$, wobei D für eine Menge an Kundenattributen steht, zum Beispiel das Alter, Geschlecht, der Familienstand, und so weiter – natürlich unter der Annahme, dass wir Zugriff auf derlei Informationen besitzen. Handelt es sich um einen Buchhändler statt um einen Supermarkt, können Produkte Bücher oder Autoren sein. Im Fall eines Web-Portals könnten Links der Gegenstand des Interesses sein, und wir können schätzen, welche Links ein Nutzer höchstwahrscheinlich anklicken wird und diese Information nutzen, um entsprechende Seiten im Voraus für einen schnelleren Zugriff zu laden.

1.2.2 Klassifikation

Unter Kredit verstehen wir den Geldbetrag, den ein Finanzinstitut, beispielsweise eine Bank, verleiht und der dann mit Zinsen und in der Regel in Raten zurückzuzahlen ist. Für die Bank ist es wichtig, im Voraus das mit einem Darlehen verbundene Risiko vorhersagen zu können, das heißt, die Wahrscheinlichkeit dafür, dass der Kunde in Verzug gerät und nicht den gesamten Betrag zurückzahlt. Dies geschieht sowohl um sicherzustellen, dass die Bank Gewinn erzielt als auch, um dem Kunden Unannehmlichkeiten durch ein Darlehen jenseits seiner finanziellen Möglichkeiten zu ersparen.

Beim sogenannten *Credit Scoring* (Hand 1998), also bei der Überprüfung der Kreditwürdigkeit von Kunden, kalkuliert die Bank das Risiko für den angefragten Kreditbetrag anhand der erhobenen Kundeninformationen. Diese Kundeninformationen beinhalten Daten, die relevant für die Berechnung der finanziellen Leistungsfähigkeit des Kunden sind – beispielsweise Einkommen, Ersparnisse, Sicherheiten, Beruf, Alter, finanzielle Vorgeschichte und so weiter. Die Bank besitzt Aufzeichnungen über vergangene Darlehen, welche Informationen über den Kunden und Aussagen dazu beinhalten, ob das jeweilige Darlehen zurückgezahlt wurde. Das Ziel besteht nun darin, anhand solcher Daten zu speziellen Darlehensanträgen auf eine allgemeingültige Regel rückzuschließen, welche die Assoziation zwischen den Attributen eines Kunden und dem zugehörigen Risiko codiert. Das heißt, das maschinelle Lernsystem passt ein Modell auf die Vergangenheitswerte an, um dadurch in der Lage zu sein, das Risiko für einen neuen Kreditantrag zu kalkulieren und entscheidet dann entsprechend, ob dieses eingegangen werden kann oder nicht.

Das obige Szenario ist ein Beispiel für ein *Klassifikationsproblem*, bei dem es zwei Klassen gibt: Kunden mit niedrigem Risiko und Kunden mit hohem Risiko. Die Informationen zu einem Kunden bilden die *Eingabe* für den Klassifikator, dessen Aufgabe darin besteht, die Eingabe einer von zwei Klassen zuzuweisen.

KLASSIFIKATION

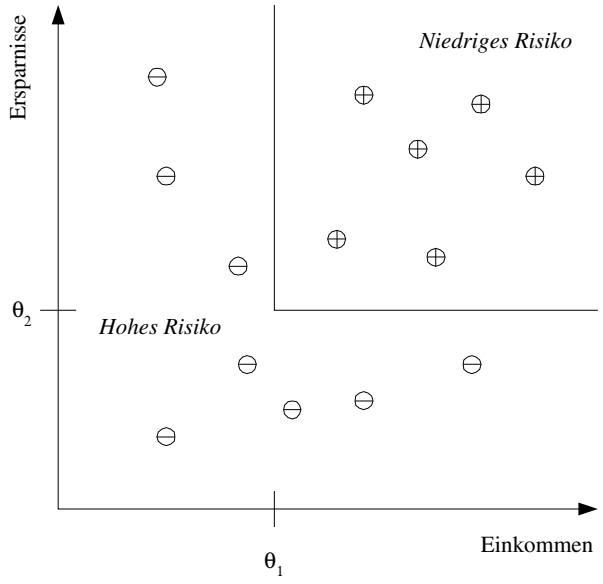


***Abb. 1.1*:** Beispiel für einen Trainingsdatensatz. Jeder Kreis repräsentiert eine Dateninstanz mit Eingabewerten an den zugehörigen Achsen. Sein Vorzeichen gibt die Klasse an. Der Einfachheit halber sind nur zwei Kundenattribute – Einkommen und Ersparnisse – in der Eingabe enthalten. Die beiden Klassen unterscheiden niedriges Risiko („+“) und hohes Risiko („−“). Eine exemplarische Diskriminanzfunktion, welche die beiden Typen von Instanzen trennt, ist ebenfalls eingefügt.

Nach dem Training mit Daten aus der Vergangenheit kann eine erlernte Klassifikationsregel für geeignete Parameterwerte θ_1 und $\theta 2$ die folgende Form haben (siehe Abbildung 1.1):

IF Einkommen$> \theta_1$ AND Ersparnisse$> \theta_2$
THEN niedriges Risiko ELSE hohes Risiko.

DISKRIMINANZFUNKTION

Dies ist ein Beispiel für eine *Diskriminanzfunktion*. Dabei handelt es sich um eine Funktion, welche die Beispiele aus unterschiedlichen Klassen voneinander trennt.

PRÄDIKTION

Die Hauptanwendung einer solchen gelernten Regel besteht in der Prädiktion. Das bedeutet, sobald wir eine Regel haben, welche auf die Vergangenheitsdaten passt, und unter der Annahme, dass die nahe Zukunft der Vergangenheit ähnelt, können wir korrekte Vorhersagen für neue Instanzen treffen. Für einen neuen Darlehensantrag und bekanntem Einkommen und Ersparnissen können wir nun recht einfach entscheiden, ob es sich um einen Fall mit niedrigem oder hohem Risiko handelt.

In manchen Fällen erscheint es uns möglicherweise sinnvoll, statt eine Entscheidung vom Typ 0/1 (niedriges Risiko/hohes Risiko) zu treffen, besser die Wahrscheinlichkeit $P(Y|X)$ zu berechnen, wobei X für die Kundenattribute steht und Y jeweils den Wert 0 bzw. 1 für die Klasse mit niedrigem Risiko bzw. mit hohem Risiko annimmt. Aus dieser Perspektive können wir die Klassifikation als das Erlernen einer Assoziation von X nach Y betrachten. Dann gilt für $X = x$, wenn $P(Y = 1|X = x) = 0{,}8$, dass bei dem entsprechenden Kunden mit 80-prozentiger Wahrscheinlichkeit ein hohes Risiko besteht – oder äquivalent dazu, dass mit 20-prozentiger Wahrscheinlichkeit ein niedriges Risiko vorliegt. Dann entscheiden wir, ob das Darlehen gewährt oder verwehrt werden soll, in Abhängigkeit vom möglichen Gewinn oder Verlust.

Mustererkennung

Im Bereich der *Mustererkennung* gibt es viele Anwendungsmöglichkeiten für das maschinelle Lernen. Ein Beispiel ist die *optische Schriftzeichenerkennung*, bei der Schriftzeichencodes anhand ihrer Abbilder erkannt werden. In diesem Fall gibt es mehrere Klassen – genauso viele nämlich, wie es Schriftzeichen gibt, die wir erkennen wollen. Von besonderem Interesse sind hier handgeschriebene Zeichen. Jeder Mensch hat seine eigene Handschrift; Zeichen können klein oder groß geschrieben werden, zu einer Seite geneigt, mit Füller oder mit Kugelschreiber – kurz gesagt, es gibt zahlreiche bildliche Darstellungen ein und desselben Schriftzeichens. Zwar ist das Schreiben eine vom Menschen erfundene Kunst, jedoch gibt es noch kein System, welches genauso akkurat ist wie ein menschlicher Leser. Wir haben keinerlei formale Definition eines Zeichens „A“, welche zum einen alle Erscheinungsformen von „A“ umfasst, zum anderen aber keine der Formen, in denen „A“ nicht auftritt. In Ermangelung solch einer Definition greifen wir auf handschriftliche Stichproben (von Schreibern) zurück und lernen anhand dieser Stichproben eine Definition dessen, was ein „A“ ausmacht. Obwohl wir also nicht genau wissen, was aus einem Abbild ein „A“ macht, sind wir jedoch sicher, dass all diese verschiedenen Darstellungen von „A“ etwas gemeinsam haben, und genau das ist es, was wir aus den Stichproben extrahieren wollen. Wir wissen, dass ein Abbild eines Schriftzeichens nicht nur eine Ansammlung von zufälligen Bildpunkten ist, sondern eine Kombination von Strichen mit einer Regelmäßigkeit, die wir mit Hilfe eines Lernprogramms erfassen können.

Wenn wir einen Text lesen, dann können wir uns unter anderem den Faktor der Redundanz in menschlichen Sprachen zunutze machen. Ein Wort ist eine *Sequenz* von Schriftzeichen, und aufeinander folgende Schriftzeichen sind nicht unabhängig voneinander, sondern aufgrund der Wörter der Sprache eingeschränkt. Das hat den Vorteil, dass wir, selbst wenn wir ein einzelnes Schriftzeichen nicht erkennen können, immer noch in der Lage sind, d?s Wort zu lesen. Derlei kontextuelle Abhängigkeiten können durchaus auch auf höherer Ebene auftauchen, zwischen Wörtern und Sätzen, durch die Syntax und Semantik der Sprache. Es gibt Algorithmen für das maschinelle Lernen, mit deren Hilfe man Sequenzen erlernen und solche Abhängigkeiten modellieren kann.

Im Falle der *Gesichtserkennung* besteht die Eingabe aus einem Bild, die Klassen sind zu erkennende Personen, und das Lernprogramm sollte lernen, die Gesichtsabbilder bestimmten Identitäten zuzuordnen. Diese Aufgabenstellung ist schwieriger als die optische Schriftzeichenerkennung, da es mehr Klassen gibt, das Eingabebild größer und ein Gesicht ein dreidimensionales Objekt ist, dessen Abbild sich durch Unterschiede in Haltung und Beleuchtung signifikant verändern kann. Außerdem ist es durchaus möglich, dass bestimmte Teile des Eingabebilds nicht sichtbar sind; zum Beispiel könnte eine Brille die Augen und Augenbrauen verdecken und das Kinn könnte unter einem Bart versteckt sein.

Bei der *medizinischen Diagnostik* werden als Eingabe die relevanten Informationen über einen Patienten genommen und als Klassen haben wir hier die Krankheiten. Die Eingabewerte beinhalten das Alter des Patienten, das Geschlecht, die medizinische Vorgeschichte und die gegenwärtigen Symptome. Einige Tests wurden unter Umständen nicht mit dem Patienten durchgeführt und daher könnten bestimmte Eingabewerte fehlen. Tests sind zeitaufwendig, können teuer sein, und bereiten dem Patienten möglicherweise Unbehagen, so dass wir sie nur dann ausführen wollen, wenn wir wirklich sicher sind, dass sie uns wertvolle Informationen liefern können. Im Fall der medizinischen Diagnostik kann eine falsche Entscheidung zu einer falschen oder gar keiner Behandlung führen, und im Zweifelsfalle ist es vorzuziehen, dass der Klassifikator die Entscheidung ablehnt oder an einen menschlichen Experten verweist.

Bei der *Spracherkennung* besteht die Eingabe aus akustischen Lauten und die Klassen sind Wörter, die geäußert werden können. Hier verläuft die zu erlernende Assoziation von einem akustischen Signal zu einem Wort irgendeiner Sprache. Bedingt durch Unterschiede in Alter, Geschlecht oder Akzent sprechen unterschiedliche Menschen dasselbe Wort unterschiedlich aus, wodurch unsere Aufgabe umso schwieriger wird. Ein weiterer Unterschied bei der Sprache liegt darin, dass die Eingabe *temporal* erfolgt; Wörter werden als zeitliche Abfolge von Sprachphonemen geäußert und manche Wörter sind länger als andere.

Akustische Informationen helfen uns nur bis zu einem gewissen Punkt, und wie bei der optischen Zeichenerkennung ist die Integration eines „Sprachmodells“ bei der Spracherkennung kritisch. Der beste Weg, um zu einem Sprachmodell zu gelangen, ist wieder der, dieses aus einem großen Korpus von Beispieldaten zu lernen. Die Anwendung des maschinellen Lernens auf die natürliche Sprachverarbeitung entwickelt sich stetig weiter. Das Filtern von Spam ist eine Anwendung, bei der die Spam-Generatoren auf der einen Seite und die Spam-Filter auf der anderen immer ausgeklügeltere Methoden finden, um einander zu übertrumpfen. Das Anfertigen von Auszügen für große Dokumente ist ein anderes interessantes Beispiel, und noch ein weiteres ist das Analysieren von Blogs oder Posts, um „Trending Topics“ zu identifizieren oder um zu entscheiden, welche Anzeigen geschaltet werden sollen. Die beeindruckendste Anwendung könnte

das maschinelle Übersetzen sein. Nach Jahrzehnten der Forschung mit händisch codierten Übersetzungsregeln sieht es heute danach aus, dass der aussichtsreichste Ansatz darin besteht, eine sehr große Anzahl von Textpaaren bereitzustellen und ein Programm automatisch die Regeln für das Abbilden von der Quellsprache in die Zielsprache herausfinden zu lassen.

Die *Biometrie* befasst sich mit der Erkennung oder Authentifizierung von Personen anhand von physiologischen und/oder verhaltensbasierten Merkmalen, wobei eine Integration von Eingaben unterschiedlicher Art notwendig ist. Beispiele für physiologische Charakteristika sind Bilder des Gesichts, Fingerabdrücke, die Iris und die Handgeometrie. Verhaltensbasierte Merkmale sind u. a. die Dynamik von Unterschrift, Stimme, Gang und Tastaturanschlag. Wenn es viele verschiedene (unkorrelierte) Eingaben gibt, sind Fälschungen (Manipulationen) im Vergleich zu gewöhnlichen Identifikationsmethoden – Foto, gedruckte Unterschrift oder Passwort – schwieriger und das System wird genauer. Idealerweise bereites es den Nutzern dabei nur wenig Unannehmlichkeiten. Maschinelles Lernen wird sowohl bei den einzelnen Erkennungsfunktionen dieser unterschiedlichen Ansätze benutzt, als auch bei der Kombination ihrer Entscheidungen, um aus diesen unter Beachtung der Verlässlichkeit der verschiedenen Quellen eine Gesamtentscheidung zu treffen.

Wissens-extraktion

Das Erlernen einer Regel anhand von Daten erlaubt ebenfalls die *Extraktion von Wissen*. Eine Regel ist ein einfaches Modell, welches die Daten erklärt, und bei genauer Betrachtung dieses Modells erhalten wir eine Erklärung für den Prozess, der den Daten zugrundeliegt. Sobald wir zum Beispiel die Diskriminante, welche die Kunden mit hohem Risiko von denen mit niedrigem Risiko trennt, erlernt haben, verfügen wir auch über die Kenntnis der Eigenschaften von Kunden mit niedrigem Risiko. Wir können diese Information dann nutzen, um potenzielle Kunden mit niedrigem Risiko effizienter in die Zielgruppe einzubinden, beispielsweise durch Werbung. Beim Lernen wird außerdem eine *Kompression* durchgeführt, indem wir durch die Anpassung einer Regel an die Daten eine Erklärung erhalten, die simpler ist als die Daten selbst, die somit weniger Speicherplatz und weniger Rechenaufwand zur Verarbeitung benötigt. Hat man einmal die Additionsregeln erlernt, muss man nicht mehr die Summe jedes möglichen Paares zweier Zahlen auswendig lernen.

Kompression

Ausreisser-erkennung

Ein weiterer Anwendungsbereich des maschinellen Lernens findet sich in der *Erkennung von Ausreißern*. Dabei sollen Instanzen gefunden werden, die nicht der Regel folgen und somit Ausnahmen darstellen. Die Idee ist, dass typische Instanzen Merkmale teilen, die einfach festzulegen sind, und Instanzen, die diese Merkmale nicht aufweisen, sind atypisch. In diesem Fall sind wir an einer Regel interessiert, die so einfach wie möglich ist und gleichzeitig einen möglichst großen Anteil unserer typischen Instanzen abdeckt. Jede Instanz, die nicht abgedeckt wird, stellt eine Ausnahme dar. Dabei kann es sich um eine Anomalie handeln, die Aufmerksam-

keit erfordert (etwa einen Betrugsversuch), oder um einen neuartigen, zuvor noch nie gesehenen, aber gültigen Fall, weshalb man auch von *Neuheitserkennung* spricht.

Neuheits-erkennung

1.2.3 Regression

Angenommen, wir benötigen ein System, welches in der Lage ist, den Preis eines gebrauchten Autos vorherzusagen. Als Eingabedaten nutzen wir all jene Attribute des Autos – Marke, Baujahr, Motorleistung, Kilometerstand und weitere Informationen – von denen wir glauben, dass sie den Wert des Autos beeinflussen. Als Ausgabe erhalten wir den Preis des Autos. Problemstellungen wie diese, bei denen die Ausgabedaten Zahlwerte sind, nennt man *Regressionsprobleme*.

Regression

Sei X die Bezeichnung der Attribute des Autos, und sei Y der Preis des Fahrzeugs. Indem wir wieder die vergangenen Transaktionen untersuchen, können wir Trainingsdaten sammeln, woraufhin das maschinelle Lernprogramm eine Funktion an diese Daten anpasst, um Y als Funktion von X zu erlernen. Abbildung 1.2 zeigt ein Beispiel, bei dem die angepasste Funktion die Form

$$y = wx + w_0$$

für geeignete Parameterwerte von w und w_0 annimmt.

Sowohl die Regression als auch die Klassifikation gehören zum Problemfeld des *überwachten Lernens*, bei dem es eine Eingabe X und eine Ausgabe Y gibt und die Aufgabe darin besteht, die Abbildung von der Eingabe auf die Ausgabe zu erlernen. Der Ansatz beim maschinellen Lernen besteht darin, dass wir von einer Modelldefinition mit einer Menge von Parametern ausgehen:

überwachtes Lernen

$$y = g(x|\theta),$$

wobei $g(\cdot)$ das Modell ist und θ seine Parameter. Y ist ein Zahlwert bei der Regression und ein Klassencode (z.B. 0/1) im Falle einer Klassifikation. $g(\cdot)$ ist die Regressionsfunktion beziehungsweise die Diskriminanzfunktion bei einer Klassifikation, welche die Instanzen der verschiedenen Klassen voneinander abgrenzt. Das maschinelle Lernprogramm optimiert die Parameter θ, so dass der Näherungsfehler minimiert wird, das heißt, unsere Schätzwerte sind so nah an den korrekten Werten aus den vorhandenen Trainingsdaten wie möglich. In Abbildung 1.2 beispielsweise ist das Modell linear und w und w_0 sind die für eine bestmögliche Anpassung an die Trainingsdaten optimierten Parameter. In Fällen, bei denen das lineare Modell zu restriktiv ist, kann man zum Beispiel eine quadratische Funktion der Form

$$y = w_2x^2 + w_1x + w_0$$

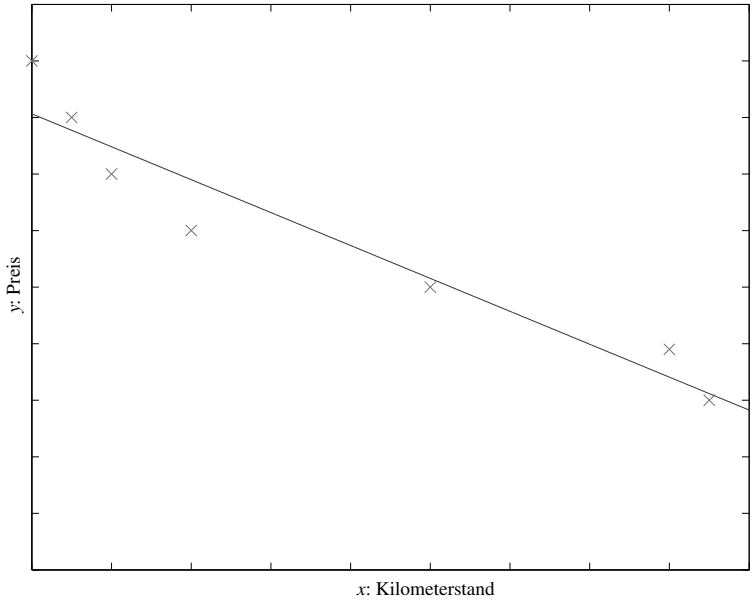


***Abb. 1.2*:** Dargestellt ist ein Trainingsdatensatz von Gebrauchtwagen und die angepasste Funktion. Aus Gründen der Einfachheit wird der Kilometerstand als einziges Eingabeattribut gewählt und ein lineares Modell benutzt.

oder eine Polynomfunktion höherer Ordnung oder eine beliebige andere nichtlineare Funktion der Eingabewerte nutzen, und in diesem Fall deren Parameter für die bestmögliche Anpassung optimieren.

Ein weiteres Beispiel für die Regression ist die Steuerung eines mobilen Roboters, beispielsweise eines autonomen Fahrzeugs, wenn der Ausgabewert aus dem Winkel besteht, um den das Lenkrad jeweils gedreht werden soll, um das Fahrzeug fortzubewegen ohne mit Hindernissen zu kollidieren oder von der Strecke abzukommen. Die Eingabedaten in Szenarien wie diesem werden durch Sensoren am Auto erzeugt, zum Beispiel einer Videokamera, einem GPS-System und so weiter. Trainingsdaten können gesammelt werden, indem die Aktionen eines menschlichen Fahrers überwacht und aufgezeichnet werden.

Wir können uns weitere Anwendungen für die Regression vorstellen, bei denen versucht wird, eine Funktion zu optimieren.[1] Angenommen, wir wollen eine Maschine bauen, die Kaffee röstet. Die Maschine hat viele Eingabewerte, welche die Qualität beeinflussen: diverse Temperatur- und Zeiteinstellungen, verschiedene Kaffeebohnensorten und so weiter. Wir führen eine Anzahl von Experimenten durch, und für verschiedene Einstellungen der Eingabewerte messen wir die Kaffeequalität, beispielsweise

[1] Mein Dank gilt Michael Jordan für dieses Beispiel.

anhand der Kundenzufriedenheit. Um die optimalen Einstellungen zu finden, passen wir ein Regressionsmodell an, welches diese Eingabewerte mit der resultierenden Kaffeequalität verbindet. Dann wählen wir neue Punkte nahe des Optimums des gegenwärtigen Modells als neue Eingabewerte, um eine bessere Konfiguration zu finden. Für diese Eingaben bestimmen wir die Qualität und fügen diese Werte zu den Daten hinzu und trainieren ein neues Modell. Bei diesem Vorgehen spricht man im Allgemeinen von einem *Response-Surface-Design*.

Manchmal wollen wir, anstatt einen absoluten numerischen Wert zu schätzen, die Möglichkeit haben, relative Positionen zu lernen. In einem Empfehlungssystem für Filme wollen wir zum Beispiel eine Liste generieren, die danach geordnet ist, für wie wahrscheinlich wir es halten, dass dem Nutzer die Filme gefallen. In Abhängigkeit von den Attributen des Films wie Genre und Schauspieler sowie den Ratings der Filme, die der Nutzer schon gesehen hat, wären wir dann in der Lage, eine *Ranking-Funktion* zu lernen, die wir verwenden können, um Empfehlungen aus neuen Filmen auszuwählen.

RANKING

1.2.4 Unüberwachtes Lernen

Beim überwachten Lernen besteht das Ziel darin, eine Abbildung von Eingabe- auf Ausgabewerte zu erlernen, wobei die korrekten Ausgabewerte bekannt sind. Beim unüberwachten Lernen sind die Ausgabewerte nicht bekannt, und uns stehen nur die Eingabedaten zur Verfügung. Das Ziel besteht darin, Regelmäßigkeiten in den Eingabewerten aufzuspüren. Es existiert eine Struktur im Eingaberaum, so dass bestimmte Muster öfter auftreten als andere, und wir wollen erkennen, was grundsätzlich passiert und was nicht. In der Statistik nennt man dies *Dichteschätzung*.

DICHTESCHÄTZUNG

Eine Möglichkeit zur Dichteschätzung ist die *Clusteranalyse*, wobei das Ziel darin besteht, Cluster (Häufungen) oder Gruppierungen von Eingabewerten zu finden. Für das Beispiel einer Firma mit Daten über ihre ehemaligen Kunden enthalten diese Kundendaten demographische Informationen sowie die vergangenen Transaktionen zwischen den betroffenen Kunden und der Firma. Letztere interessiert sich nun unter Umständen für das Profil ihrer Kunden, um zu erkennen, welche Art Kunde häufig auftritt. In solch einem Fall weist ein Clustermodell alle Kunden mit ähnlichen Attributen derselben Gruppe zu und stellt damit der Firma eine natürliche Gruppierung ihrer Kunden zur Verfügung; dies wird als *Kundensegmentierung* bezeichnet. Sobald solche Gruppen gefunden wurden, kann die Firma gruppenspezifische Strategieentscheidungen treffen, zum Beispiel hinsichtlich ihrer Dienstleistungen und Produkte; dies ist unter dem Begriff *Kundenbeziehungsmanagement* bekannt. Solche Gruppierungen erlauben ebenfalls die Identifikation von Ausreißern, sprich Kunden, die sich von anderen Kunden unterscheiden, was wiederum eine Marktnische für eine mögliche Expansion der Firma aufzeigen könnte.

CLUSTERANALYSE

Ein interessantes Anwendungsfeld für die Clusteranalyse ist die *Bildkompression*. Hierbei bestehen die Eingabeinstanzen aus Bildpunkten, die in Form von RGB-Werten dargestellt sind. Ein Clusteringprogramm gruppiert Pixel mit ähnlichen Farben in denselben Gruppen, und diese Pixelgruppen korrespondieren zu den Farben, die häufig im Bild vorkommen. Wenn in einem Bild nur Schattierungen von wenigen Farben vorhanden sind, und wenn wir alle einer Gruppe zugehörigen Schattierungen mit derselben Farbe codieren, zum Beispiel mit ihrem Durchschnittswert, dann wird das Bild quantisiert. Gehen wir einmal davon aus, dass die Pixel eine Größe von 24 Bit haben, um 16 Millionen Farben zu repräsentieren, aber dass es nur Schattierungen von 64 Hauptfarben gibt; wir benötigen daher für jedes Pixel nur 6 Bit statt 24. Wenn zum Beispiel eine Grafik verschiedene Nuancen der Farbe Blau in unterschiedlichen Bildregionen aufweist, und wenn wir denselben Durchschnittswert für all die Blautöne nutzen, verlieren wir zwar Details im Bild, gewinnen aber an Speicher- und Datenübertragungskapazität. Im Idealfall möchte man daher durch die Analyse von sich wiederholenden Bildmustern übergeordnete Regelmäßigkeiten finden, also zum Beispiel Texturen, Objekte, etc. Dadurch ermöglicht sich auf einer höheren Ebene eine einfachere und nützlichere Beschreibung der Szenerie, und es wird beispielsweise eine bessere Kompressionsrate erreicht als bei Kompression auf Pixelebene. Im Falle von abgelichteten Dokumentseiten haben wir nicht einfach zufällig ein Pixel hier und dort, sondern Bitmap-Bilder von Schriftzeichen. Es gibt eine Struktur in den Daten, und wir nutzen einfach diese Redundanz aus, indem wir eine kürzere Beschreibung der Daten finden: Ein 16 x 16 Bitmap von „A“ benötigt 32 Bytes, seine ASCII-Codierung jedoch lediglich 1 Byte.

Beim *Clustern von Dokumenten* besteht das Ziel darin, die Dokumente anhand von Ähnlichkeiten in Gruppen anzuordnen. Beispielsweise können Nachrichten weiter unterteilt werden in solche, die aus den Bereichen Politik, Sport, Mode, Kultur und weiteren stammen. Üblicherweise werden Dokumente als *Bag-of-Words* repräsentiert. Dabei verwenden wir ein vordefiniertes Lexikon aus N Wörtern, und jedes Dokument ist ein N-dimensionaler binärer Vektor, dessen i-te Komponente 1 ist, falls das Wort i in dem Dokument vorkommt. Suffixe werden entfernt, um Dubletten zu vermeiden; ebenso werden Wörter wie „und“, „von“ usw. nicht mit verwendet, da sie keine Information tragen. Die Dokumente werden dann anhand der Anzahl gemeinsamer Wörter gruppiert. Das Ergebnis hängt natürlich kritisch von dem verwendeten Lexikon ab.

Die Methoden des maschinellen Lernens werden auch in der *Bioinformatik* verwendet. Die DNS in unserem Genom ist die „Blaupause des Lebens“ und besteht aus einer Sequenz von Basen, nämlich A, G, C und T. RNS wird aus der DNS transkribiert und aufgrund der RNS werden Proteine gebildet. Proteine sind quasi das, was der lebende Organismus ist und tut. Genauso wie DNS eine Sequenz von Basen ist, besteht ein Protein aus einer Sequenz von Aminosäuren (definiert durch die Basen). Ein

Anwendungsgebiet für die Informatik in der Molekularbiologie findet sich dabei in der Suche nach *übereinstimmenden Sequenzen*. Dabei handelt es sich um ein sehr kompliziertes Stringvergleichsproblem, denn die Strings können sehr lang sein, es gibt viele Vorlagenstrings, die es abzugleichen gilt, und es können Löschungen, Einfügungen und Ersetzungen auftreten. Die Clusteranalyse wird dabei zum Erlernen von *Motiven* genutzt, wobei es sich um Sequenzen von Aminosäuren handelt, die wiederholt in Proteinen auftreten. Motive sind von Interesse, da sie strukturellen oder funktionalen Elementen innerhalb der Sequenzen, die sie charakterisieren, entsprechen können. Die Analogie liegt hierbei darin, dass Aminosäuren wie Buchstaben und Proteine wie Sätze betrachet werden; Motive sind wie Wörter, sprich ein String aus Buchstaben mit einer bestimmten Bedeutung, der oft in verschiedenen Sätzen vorkommt.

1.2.5 Bestärkendes Lernen

In einigen Anwendungen besteht die Ausgabe eines Systems aus *Aktionen*. In so einem Fall ist eine einzelne Aktion nicht wichtig; von Bedeutung ist jedoch die *Taktik*, sprich die Sequenz von korrekten Aktionen, die zum Ziel führen. So etwas wie die beste Aktion in einem beliebigen Zwischenstadium gibt es nicht; eine Aktion ist erst dann gut, wenn sie Teil einer guten Taktik ist. In solch einem Szenario sollte das maschinelle Lernprogramm in der Lage sein, die Güte von Taktiken einzuschätzen und von vergangenen guten Aktionssequenzen zu lernen, um eine Taktik generieren zu können. Derartige Lernmethoden nennt man Algorithmen des *bestärkenden Lernens*.

BESTÄRKENDES LERNEN

Ein passendes Beispiel hierfür bieten *Brettspiele*, bei denen ein einzelner Zug selbst nicht so wichtig ist; entscheidend ist jedoch die Sequenz von richtigen Zügen. Ein Zug ist gut, wenn er Teil einer guten Spieltaktik ist. Das Spielen bildet ein wichtiges Forschungsgebiet sowohl im Bereich der künstlichen Intelligenz als auch beim maschinellen Lernen. Dies liegt vor allem daran, dass es zwar leicht ist, Spiele zu beschreiben, aber gleichzeitig recht schwierig, sie erfolgreich zu spielen. Ein Brettspiel wie Schach hat nur eine geringe Zahl von Regeln, aber seine Komplexität erlangt es durch die große Anzahl an möglichen Zügen in jeder Situation und die schiere Menge an Zügen, die ein Spiel beinhaltet. Sobald wir einen guten Algorithmus haben, der lernen kann, Spiele erfolgreich zu spielen, können wir ihn ebenfalls für Anwendungen mit offensichtlicherem ökonomischem Nutzen einsetzen.

Ein Roboter, der sich in einer bestimmten Umgebung auf der Suche nach einem Zielpunkt bewegt, zeigt ein weiteres Anwendungsfeld des bestärkenden Lernens auf. Zu jeder Zeit kann der Roboter sich in eine von mehreren Richtungen bewegen. Nach einer Reihe von Versuchen sollte er die korrekte Sequenz von Aktionen erlernen, mit denen er den Zielzustand von einem Ausgangszustand aus erreichen kann, und zwar schnellstmöglich und ohne irgendeines der Hindernisse zu treffen.

Ein Umstand, der bestärkendes Lernen schwieriger gestaltet, entsteht dann, wenn das System unverlässliche oder nur Teile der Sensorinformationen zur Verfügung hat. Ein mit einer Kamera ausgestatteter Roboter zum Beispiel hat nur unvollständige Informationen zur Verfügung und befindet sich daher ständig in einem nur *teilweise beobachtbaren Zustand*. Folglich sollte er seine Entscheidungen unter Beachtung dieser Unsicherheit treffen. Eine Aufgabe könnte auch die gleichzeitige Arbeit von *mehreren Agenten* erfordern, welche interagieren und kooperieren sollten, um ein gemeinsames Ziel zu erreichen. Ein Beispiel hierfür ist ein Team von Fußball spielenden Robotern.

1.3 Anmerkungen

Die Evolution ist die treibende Kraft, die unseren Körperbau sowie die uns angeborenen Instinkte und Reflexe bestimmt. Wir lernen auch, unser Verhalten während unseres Lebens zu ändern. Das hilft uns z.B., mit Umweltveränderungen fertig zu werden, die nicht durch die Evolution vorhergesagt werden können. Organismen mit kürzerer Lebenszeit in einer gut definierten Umwelt mögen all ihre Verhaltensweisen bei der Geburt mitbekommen, aber anstatt alle möglichen Verhaltensweisen für jeden beliebigen Umstand fest vorzugeben, dem wir im Leben begegnen könnten, stattete uns die Evolution mit einem großen Gehirn und Lernfähigkeit aus, so dass wir uns selbständig anhand unserer Erfahrungen an verschiedene Umgebungen anpassen können. Wenn wir die beste Strategie in einer bestimmten Situation lernen, wird dieses Wissen in unserem Gehirn gespeichert, und sollte die bestimmte Situation wieder auftreten, können wir – wenn sie uns kognitiv vertraut ist („kognitiv“ kommt dabei vom lateinischen *cognoscere* für „erkennen“) – die geeignete Strategie abrufen und entsprechend handeln. Das Lernen hat jedoch auch seine Grenzen; es gibt sicherlich Dinge, die wir mit der beschränkten Kapazität unseres Gehirns nie lernen werden können, genauso wenig, wie wir nie werden „lernen“ können, uns einen dritten Arm wachsen zu lassen oder ein Auge auf der Rückseite unseres Kopfes, selbst wenn beides noch so nützlich wäre. Man beachte, dass im Gegensatz zur Psychologie, der kognitiven Wissenschaft oder der Neurowissenschaft unser Ziel beim maschinellen Lernen nicht darin besteht, den Prozess, der dem Lernen bei Mensch und Tier zugrunde liegt, zu verstehen, sondern nutzbare Systeme zu bauen, so wie in jedem anderen Bereich des Ingenieurwesens auch.

Fast die gesamte Wissenschaft besteht aus nichts anderem als dem Anpassen von Modellen an erhobene Daten. Wissenschaftler entwerfen Experimente, führen Beobachtungen durch und sammeln Daten. Dann versuchen sie, Wissen aus den Daten zu extrahieren, indem sie einfache Modelle finden, welche die beobachteten Daten erklären. Dieses Vorgehen nennt man *Induktion* und meint damit das Ableiten allgemeiner Regeln aus einer Reihe von speziellen Fällen.

Wir sind nun an einem Punkt angelangt, an dem solche Datenanalysen nicht länger von Menschen vorgenommen werden können, da zum einen die Menge an Daten riesig ist und zum anderen qualifizierte Kräfte für solche Analysen rar und manuelle Analysen kostspielig sind. Daher besteht wachsendes Interesse an Computermodellen, die Daten analysieren und automatisch Informationen aus ihnen extrahieren können, sprich, die „lernen“ können.

Die Methoden, welche wir in den folgenden Kapiteln diskutieren werden, haben ihren Ursprung in verschiedenen wissenschaftlichen Domänen. Manchmal wurde derselbe Algorithmus unabhängig voneinander in mehr als einem Gebiet und demnach unterschiedlichen historischen Pfaden folgend entwickelt.

In der Statistik wird mit *Inferenz* das Rückschließen von speziellen Beobachtungen auf allgemeine Beschreibungen bezeichnet und das Lernen nennt man *Schätzung*. Die Klassifikation bezeichnet man in der Statistik als *Diskriminanzanalyse* (McLachlan 1992; Hastie, Tibshirani und Friedman 2011). Bevor Computer erschwinglich und allgegenwärtig wurden, konnten Statistiker nur mit kleinen Stichproben arbeiten. Statistiker nutzten, als Mathematiker, die sie sind, zumeist einfache parametrische Modelle, die mathematisch analysiert werden konnten. Im Ingenieurswesen nennt man die Klassifikation *Mustererkennung* und es handelt sich um einen nicht-parametrischen und viel empirischeren Ansatz (Duda, Hart und Stork 2001; Webb und Copey 2011).

Das maschinelle Lernen ist mit dem Gebiet der *künstlichen Intelligenz* (Russell und Norvig 2009) verwandt, da ein intelligentes System in der Lage sein sollte, sich an Veränderungen seiner Umwelt anzupassen. Aufgaben wie das Sehen, die Sprache und die Robotik sollten am besten ebenfalls anhand von Stichprobendaten erlernt werden. In der Elektrotechnik resultierte Forschung im Bereich der *Signalverarbeitung* in adaptiven Computervisions- und Sprachprogrammen. Unter ihnen ist die Entwicklung von *Hidden-Markov-Modellen* (HMM) für die Spracherkennung von besonderer Bedeutung.

Durch Fortschritte in der VLSI-Technologie (also in der Entwicklung hochintegrierter Systeme; engl. *Very Large Scale Integration*) und der Möglichkeit, parallele Hardware zu bauen, die Tausende von Prozessoren beinhaltet, wurde in den späten 1980er Jahren das Feld der *künstlichen neuronalen Netze* wiederbelebt als eine mögliche Theorie für die Verteilung von Rechenleistung auf eine große Zahl von Prozessoreinheiten (Bishop 1995). Nach und nach erkannten die Anhänger der neuronalen Netze, dass die meisten Lernalgorithmen neuronaler Netze ihr Fundament in der Statistik haben – beispielsweise ist das mehrlagige Perzeptron eine weitere Klasse von nicht-parametrischen Schätzfunktionen – und Behauptungen hinsichtlich gehirnähnlicher Rechenleistung verschwanden langsam von der Bildfläche.

In den letzten Jahren haben kernelbasierte Algorithmen wie etwa Support-Vektor-Maschinen große Verbreitung erfahren. Durch die Verwendung von Kernel-Funktionen können sie an eine Vielzahl von Anwendungen angepasst werden. Diese Algorithmen werden insbesondere in der Bioinformatik und bei der Sprachverarbeitung eingesetzt. Heute gehört es zum Allgemeinwissen, dass ein gute Datenrepräsentation entscheidend für das Lernen ist, und Kernel-Funktionen haben sich als ein sehr gutes Mittel erwiesen, um Expertenwissen zu implementieren.

Ein weiterer moderner Ansatz ist die Verwendung von *generativen Modellen*. Diese erklären die beobachteten Daten über die Interaktion einer Menge von verborgenen Faktoren. Meist werden *Graphenmodelle* verwendet, um die Interaktion zwischen den Faktoren und den Daten zu visualisieren. Mit dem *Bayesschen Formalismus* können wir unsere a-priori-Information über die verborgenen Faktoren und das Modell definieren und die Modellparameter ableiten.

Inzwischen ist es dank sinkender Kosten für das Speichern und Vernetzen möglich geworden, sehr große Datenmengen über das Internet verfügbar zu halten. Zusammen mit den gleichfalls gesunkenen Kosten für Rechenleistung hat diese Entwicklung die Möglichkeit eröffnet, Lernalgorithmen auf großen Datenmengen laufen zu lassen. In den vergangenen Jahrzehnten war allgemein erwartet worden, dass für künstliche Intelligenz ein neues Paradigma nötig wäre, eine neue Herangehensweise, ein neues Modell des Rechnens oder ein völlig neuer Typ von Algorithmen.

Angesichts der jüngsten Erfolge auf vielen Gebieten des maschinellen Lernens können wir heute konstatieren, dass der entscheidende Faktor nicht die Algorithmen waren, sondern die große Menge von Beispieldaten und die entsprechend große Rechenkapazität, um Lernalgorithmen auf diesen Datenmengen laufen zu lassen. Beispielsweise gehen Support-Vektor-Maschinen auf Potentialfunktionen, lineare Klassifikatoren und nachbarbasierte Methoden zurück, die in den 1950er und 1960er Jahren vorgeschlagen wurden – es gab damals nur noch keine ausreichend schnellen Computer oder nicht genug Speicherplatz, um das Potential dieser Algorithmen zu nutzen. Es ist zu vermuten, dass Aufgaben wie das maschinelle Übersetzen und die Projektplanung von solch relativ einfachen Lernalgorithmen übernommen werden können, wobei die Algorithmen allerdings auf großen Datenmengen oder durch lange Trial-and-Error-Läufe trainiert werden müssen. Die jüngsten Erfolge mit „Deep-Learning“-Algorithmen stützen diese These. Intelligenz resultiert offenbar nicht aus einer Wunderformel, sondern aus dem beharrlichen, einer Brute-Force-Vorgehensweise nicht unähnlichen, Anwenden von einfachen, überschaubaren Algorithmen.

Data Mining ist ein im kommerziellen Bereich aufgekommener Begriff für die Anwendung von Lernalgorithmen auf große Datenmengen (Witten und Frank 2011; Han und Kamber 2011). In der Informatik bezeichnet man dies auch als *Knowledge Discovery in Datenbanken* (KDD).

Die Forschung in den verschiedenen Wissensgebieten (Statistik, Mustererkennung, neuronale Netze, Signalverarbeitung, Steuerungswesen, künstliche Intelligenz und Data Mining) folgte in der Vergangenheit diversen Ansätzen mit unterschiedlichen Schwerpunkten. Dieses Buch hat zum Ziel, all diese Schwerpunkte zusammenzuführen, um eine allumfassende Behandlung der Probleme und Lösungen zu liefern.

1.4 Relevante Ressourcen

Die aktuellsten Forschungsergebnisse zum maschinellen Lernen finden sich breit gefächert in wissenschaftlichen Magazinen und Konferenzen zu verschiedenen Themenfeldern. Magazine speziell zum Thema sind *Machine Learning* und das *Journal of Machine Learning Research*. Zeitschriften wie *Neural Computation*, *Neural Networks* und die *IEEE Transactions on Neural Networks* veröffentlichen ebenfalls viel zum maschinellen Lernen. Statistikzeitschriften wie die *Annals of Statistics* und das *Journal of the American Statistical Association* veröffentlichen Arbeiten, die aus dem Blickwinkel des maschinellen Lernens interessant sind. Viele IEEE-Publikationen wie *Transactions on Pattern Analysis and Machine Intelligence*, *Transactions on Systems, Man, and Cybernetics*, *Transactions on Image Processing* oder *Transactions on Signal Processing* enthalten interessante Beiträge sowohl zur Theorie des maschinellen Lernens als auch zu zahlreichen Anwendungen.

Magazine zur künstlichen Intelligenz, Mustererkennung, Fuzzy-Logik und Signalverarbeitung enthalten ebenfalls Forschungsberichte zum maschinellen Lernen. Zeitschriften mit Fokus auf Data Mining sind *Data Mining and Knowledge Discovery*, *IEEE Transactions on Knowledge and Data Engineering* und das *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Journal*.

Die wichtigsten Konferenzen zum maschinellen Lernen sind *Neural Information Processing Systems* (NIPS), *Uncertainty in Artificial Intelligence* (UAI), die *International Conference on Machine Learning* (ICML), die *European Conference on Machine Learning* (ECML), *Artificial Intelligence and Statistics* (AIS-TATS) und *Computational Learning Theory* (COLT). Auch in Konferenzen zu Forschungsgebieten wie Mustererkennung, neuronale Netze, künstliche Intelligenz, Fuzzy-Logik und genetische Algorithmen sowie zu Anwendungsgebieten wie Computervision, Sprachtechnologie, Robotik und Data Mining gibt es Sitzungen zum maschinellen Lernen.

Das *UCI Repository* (`http://archive.ics.uci.edu/ml`) enthält eine große Anzahl von Datenbeständen, die von Forschern auf dem Gebiet des maschinellen Lernens für Benchmark-Tests genutzt werden können. Eine andere Quelle ist das *Statlib Repository*, das unter `http://lib.stat.cmu.edu` zu finden ist. Außerdem gibt es Repositorys für spezielle An-

wendungen, beispielsweise in der Bioinformatik, für die Gesichterkennung oder die Spracherkennung.

Zu diesen Repositorys werden ständig neue und größere Datensätze hinzugefügt. Dennoch sind einige Forscher der Ansicht, dass solche Repositorys die Gesamtheit der Merkmale von realen Daten nicht vollständig widerspiegeln und nur einen begrenzten Gültigkeitsbereich haben. Die mit diesen Datensätzen erzielte Genauigkeit hätte somit keinen großen Aussagegehalt. Wenn beim Zuschnitt eines neuen Algorithmus einige Datensätze aus einem festen Repository wiederholt benutzt werden, dann konstruieren wir eine neue Menge von „UCI-Algorithmen“, die auf eben diese Datensätze spezialisiert sind. Es verhält sich ähnlich wie mit Studenten, die für eine Klausur lernen, indem sie immer nur die gleiche Menge von Testfragen beantworten. Wie wir in späteren Kapiteln sehen werden, ist es ohnehin besser, verschiedene Algorithmen für verschiedene Aufgaben zu haben. Daher ist es vorzuziehen, eine bestimmte Anwendung im Blick zu haben, für diese eine oder mehrere große Datenmengen zu erzeugen und darauf dann verschiedene Algorithmen zu vergleichen, die das betrachtete Problem lösen.

Die aktuellsten Papers zum maschinellen Lernen sind online verfügbar. Die meisten Autoren stellen auch ihre Codes und die Daten im Internet zur Verfügung. Mitschnitte von Tutorials, die auf Konferenzen und Sommerschulen gegeben wurden, sind ebenfalls größtenteils im Netz zu finden. Außerdem gibt es freie Software-Pakete, in denen verschiedene Algorithmen des maschinellen Lernens implementiert sind. Ein besonders erwähnenswertes Software-Paket ist Weka, das unter `http://www.cs.waikato.ac.nz/ml/weka/` zu finden ist.

1.5 Übungen

1. Stellen Sie sich vor, Sie haben zwei Möglichkeiten: Sie können ein Dokument faxen, also das Bild senden, oder Sie können ein optisches Schriftzeichenlesegerät (Optical Character Reader, OCR) benutzen und die Textdatei senden. Diskutieren Sie die Vor- und Nachteile beider Ansätze auf vergleichende Art. Wann wäre einer der beiden Ansätze dem anderen vorzuziehen?

2. Nehmen wir an, wir wollen ein OCR bauen und für jedes Schriftzeichen speichern wir die Bitmapdatei des Zeichens als Vorlage ab, mit der wir das abgelesene Zeichen Pixel für Pixel abgleichen. Erklären Sie, wann ein derartiges System versagen würde. Warum werden immer noch Barcode-Lesegeräte genutzt?

   Lösung: Ein solches System erlaubt nur ein Template pro Zeichen und kann zum Beispiel nicht zwischen verschiedenen Fonts ein und desselben Zeichens unterscheiden. Es gibt standardisierte Fonts wie OCR-A und OCR-B – diese sehen wir typischerweise auf den

Verpackungen der Waren, die wir kaufen –, die mit OCR-Software benutzt werden. Die Zeichen dieser Fonts wurden ein klein wenig verändert, um die Ähnlichkeiten zwischen verschiedenen Zeichen zu minimieren. Barcode-Lesegeräte werden nach wie vor eingesetzt, weil das Erkennen von Barcodes immer noch Vorteile (Kosten, Zuverlässigkeit, Verfügbarkeit) gegenüber dem Erkennen von Zeichen in beliebigen Fonts, Größen und Schriftschnitten hat.

3. Angenommen, uns wird die Aufgabe übertragen, ein System zu entwerfen, welches Spam-Mails unterscheiden kann. Was ist in Spam-Mails enthalten, das uns wissen lässt, dass es sich um Spam handelt? Wie kann der Rechner Spam mittels syntaktischer Analyse erkennen? Welche Aktion würden Sie sich von einem Rechner wünschen, wenn er eine Spam-Mail entdeckt – soll er sie automatisch löschen, sie in einen anderen Ordner verschieben oder sie nur am Bildschirm hervorheben?

   Lösung: Typischerweise prüfen textbasierte Spamfilter, ob bestimmte Wörter und Symbole vorhanden bzw. nicht vorhanden sind. Wörter wie „Gelegenheit“, „Viagra“, „Dollar“ oder auch Zeichen wie „$“ und „!“ erhöhen die Wahrscheinlichkeit, dass es sich bei einer E-Mail um Spam handelt. Diese Wahrscheinlichkeiten werden anhand einer Trainingsmenge von E-Mails gelernt, die der Nutzer zuvor als Spam gekennzeichnet hat. Wir werden in den folgenden Kapiteln viele Algorithmen für diese Aufgabe kennenlernen.

   Spamfilter arbeiten nicht zu hundert Prozent zuverlässig und machen eventuell Fehler bei der Klassifikation. Wenn eine Spam-Mail nicht herausgefiltert wird, ist das natürlich nicht gut, doch es ist nicht so schlimm wie das Herausfiltern einer erwünschten E-Mail. Wir werden später noch diskutieren, wie die relativen Kosten solcher falsch positiven und falsch negativen Klassifikationen berücksichtigt werden können.

   Wegen dieser Problematik ist es nicht ratsam, E-Mails, die von einem Filter als Spam erkannt wurden, automatisch zu löschen. Stattdessen sollten sie so abgelegt werden, dass der Nutzer die Möglichkeit hat, sie anzuschauen, falls er dies möchte. Empfehlenswert ist diese Kontrolle vor allem in der Anfangsphase der Arbeit des Spamfilters, da dieser dann noch nicht ausreichend trainiert ist. Das Filtern von Spam ist wahrscheinlich eines der besten Beispiele für lernende Systeme, die sich an Änderungen (in diesem Fall bei der Generierung von Spam) anpassen können.

4. Gehen wir davon aus, dass Sie die Aufgabe bekommen, ein automatisiertes Taxi zu bauen. Definieren Sie die Randbedingungen! Was sind die Eingabewerte? Woraus besteht die Ausgabe? Wie können Sie mit dem Fahrgast kommunizieren? Müssen Sie mit anderen automatisierten Taxis kommunizieren, sprich, brauchen Sie eine „Sprache“?

5. Bei der Warenkorbanalyse wollen wir Abhängigkeiten zwischen zwei Artikeln X und Y herausfinden. Wie können Sie bei vorliegender Datenbank mit Kundentransaktionen diese Abhängigkeiten aufspüren? Wie würden Sie dies für mehr als zwei Artikel verallgemeinern?

6. Suchen Sie aus einer Tageszeitung für jede der Kategorien Politik, Sport und Kultur fünf Beispielmeldungen heraus. Sehen Sie die Meldungen durch und identifizieren Sie für jede Kategorie Wörter, die besonders häufig verwendet werden und die dabei helfen können, zwischen den verschiedenen Kategorien zu unterscheiden. Beispielsweise wird eine Meldung der Kategorie Politik wahrscheinlich Wörter wie „Regierung“, „Rezession“ und „Parlament“ enthalten, während in einer Meldung aus der Kategorie Kultur Wörter wie „Album“, „Leinwand“ oder „Theater“ vorkommen werden. Es wird auch Wörter wie „Ball“ geben, die mehrdeutig sind.

7. Wenn ein Foto eines Gesichts von 100×100 Pixeln zeilenweise geschrieben wird, ergibt das einen Vektor mit 10 000 Komponenten. Wenn wir das Bild um einen Pixel nach rechts verschieben, erhalten wir einen völlig anderen Vektor. Wie müssen Gesichtserkenner konstruiert sein, damit sie gegenüber solchen Verschiebungen robust sind?

   Lösung: Um die Eingabedaten zu normieren, arbeiten Gesichtserkennungssysteme meist mit einer Vorverarbeitungsstufe, in der das Bild vor der Erkennung zentriert und eventuell skaliert wird. Dazu werden in der Regel zunächst die Augen identifiziert, um dann das Bild entsprechend zu verschieben. Es gibt auch Erkennungssysteme, die das Bild des Gesichts nicht als Pixelbild verwenden, sondern stattdessen strukturelle Merkmale aus dem Bild extrahieren, beispielsweise das Verhältnis zwischen dem Augenabstand und der Größe des Gesichts. Solche Merkmale sind invariant gegenüber Translationen und Skalierungen.

8. Betrachten Sie als Beispiel das Wort „Maschine“. Schreiben Sie es zehnmal auf. Bitten Sie einen Freund, es ebenfalls zehnmal aufzuschreiben. Analysieren Sie dann die zwanzig Bilder und versuchen Sie, Merkmale – typische Striche, Bögen, Schlaufen; die Art, wie Punkte gesetzt sind – zu finden, welche die Unterscheidung ihrer Handschrift von der Ihres Freundes erlauben.

9. Wenn man den Wert eines Gebrauchtwagens schätzen möchte, dann ist es sinnvoller, den prozentualen Wertverlust gegenüber dem ursprünglichen Verkaufspreis zu schätzen, anstatt den absoluten Preis. Warum?

1.6 Literaturangaben

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Duda, R. O., P. E. Hart and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Han, J., and M. Kamber. 2011. *Data Mining: Concepts and Techniques,* 3rd ed. San Francisco: Morgan Kaufmann.

Hand, D. J. 1998. „Consumer Credit and Statistics.“ In *Statistics in Finance,* ed. D. J. Hand and S. D. Jacka, 69–81. London: Arnold.

Hastie, T., R. Tibshirani and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed. New York: Springer.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Russell, S. and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach.* 3rd ed. New York: Prentice Hall.

Webb, A., and K. D. Copsey. 2011. *Statistical Pattern Recognition.* 3rd ed. New York: Wiley.

Witten, I. H., and E. Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques.* 2nd ed. San Francisco: Morgan Kaufmann.

2 Überwachtes Lernen

Wir diskutieren das überwachte Lernen, beginnend mit dem einfachsten Fall, bei dem eine Klasse anhand ihrer positiven und negativen Beispiele zu erlernen ist. Wir verallgemeinern dies und befassen uns mit dem Fall von multiplen Klassen, gefolgt von der Regression, bei der es kontinuierliche Ausgabewerte gibt.

2.1 Erlernen einer Klasse anhand von Beispielen

Nehmen wir an, wir wollen die *Klasse*, bezeichnet mit $\mathcal{C}$, für einen „Familienwagen" erlernen. Wir haben eine Menge an Beispielen für Autos und verfügen außerdem über eine Gruppe von Personen zur Befragung, denen wir diese verschiedenen Autos zeigen. Die Befragten sehen sich die Wagen, die wir ihnen zeigen, an und ordnen sie; Autos, die sie als Familienwagen ansehen, sind *positive Beispiele*, die anderen Wagen bilden die Menge der *negativen Beispiele*. Das Erlernen einer Klasse besteht darin, eine Beschreibung zu finden, die auf alle positiven und auf keins der negativen Beispiele zutrifft. Durch diese Vorgehensweise können wir dann Vorhersagen treffen. Für einen Wagen, den wir vorher noch nicht betrachtet haben, können wir durch Vergleich mit der erlernten Beschreibung sagen, ob es sich um einen Familienwagen handelt oder nicht. Wir können aber auch Wissen extrahieren: die Studie wird möglicherweise von einem Autohersteller unterstützt und das Ziel mag darin bestehen, zu verstehen, was die Leute von einem Familienwagen erwarten.

POSITIVE BEISPIELE

NEGATIVE BEISPIELE

Gehen wir nun davon aus, dass wir nach einigen Diskussionen mit Experten auf dem Gebiet die Schlussfolgerung ziehen, dass von all den Merkmalen, die ein Auto haben kann, der Preis und die Motorenleistung diejenigen sind, die einen Familienwagen von anderen Autos unterscheiden. Diese zwei Attribute bilden die *Eingaben* für den *Klassifikator*. Man beachte, dass wir bei der Festlegung dieser speziellen *Eingaberepräsentation* diverse andere Attribute als irrelevant ansehen. Zwar kommen einem sicherlich andere Attribute in den Sinn, die für die Unterscheidung zwischen Autotypen wichtig sein könnten – beispielsweise die Anzahl von Sitzplätzen oder die Farbe – wir werden jedoch nur den Preis und die Motorenleistung betrachten, um dieses Beispiel einfach zu halten.

EINGABE-REPRÄSENTATION

Nehmen wir den Preis als erstes Eingabeattribut x_1 und die Motorenleistung als zweites Eingabeattribut x_2. Dann stellen wir jedes Auto mit zwei numerischen Werten der Form

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.1}$$

dar und die zugehörige Kennung r gibt den Typ an:

$$r = \begin{cases} 1 & \text{wenn } \boldsymbol{x} \text{ ein positives Beispiel ist ,} \\ 0 & \text{wenn } \boldsymbol{x} \text{ ein negatives Beispiel ist .} \end{cases} \tag{2.2}$$

Jeder Wagen wird durch solch ein geordnetes Paar $(\boldsymbol{x}, r)$ repräsentiert und die Übungsmenge enthält N solcher Beispiele der Form

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N , \tag{2.3}$$

wobei t den Index für verschiedene Beispiele in der Menge darstellt; er steht nicht für zeitliche oder ähnliche Ordnungen.

Unsere Trainingsdaten können nun im zweidimensionalen (x_1, x_2)-Raum abgebildet werden, wobei jede Instanz t ein Datenpunkt an den Koordinaten (x_1^t, x_2^t) ist und ihr Typ, sprich positiv oder negativ, durch r^t gegeben ist (siehe Abbildung 2.1).

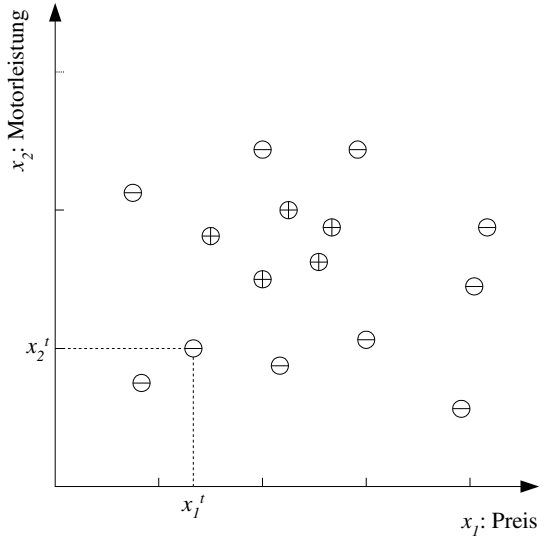


Abb. 2.1: Übungsdatenmenge für die Klasse „Familienwagen“. Jeder Datenpunkt entspricht einem Beispielwagen. Die Koordinaten des Punkts geben den Preis und die Motorenleistung des Autos an. „+“ steht für ein positives Beispiel der Klasse (also für einen Familienwagen), „-“ repräsentiert ein negatives Beispiel (keinen Familienwagen), also einen anderen Wagentyp.

Nach weiteren Diskussionen mit Experten und der Analyse der Daten haben wir nun möglicherweise Grund zur Annahme, dass es sich nur dann um einen Familienwagen handelt, wenn der Preis und die Motorenleistung in einem bestimmten Bereich

$$(p_1 \leq \text{Preis} \leq p_2) \text{ AND } (e_1 \leq \text{Motorleistung} \leq e_2) \tag{2.4}$$

mit geeigneten Werten für p_1, p_2, e_1 und e_2 liegen. Gleichung 2.4 geht somit davon aus, dass $\mathcal{C}$ ein Rechteck im durch Preis und Motorenleistung aufgespannten Raum ist (siehe Abbildung 2.2).

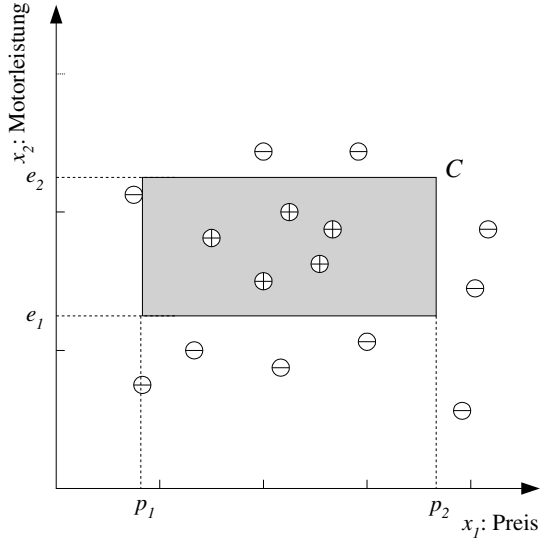


***Abb. 2.2*:** Beispiel einer Hypothesenklasse. Die Klasse Familienwagen ist ein Rechteck im durch Preis und Motorenleistung aufgespannten Raum.

Gleichung 2.4 legt die *Hypothesenklasse* $\mathcal{H}$ fest, nämlich die Menge aller Rechtecke, von der wir annehmen, dass $\mathcal{C}$ aus ihr gezogen wurde. Der Lernalgorithmus sollte dann eine spezielle *Hypothese* finden, $h \in \mathcal{H}$, um $\mathcal{C}$ so genau als möglich zu approximieren.

HYPOTHESENKLASSE

HYPOTHESE

Zwar definieren Experten die Hypothesenklasse, jedoch können sie nicht sagen, welche Werte die Parameter annehmen; mit anderen Worten, wir wählen zwar $\mathcal{H}$ aus, jedoch wissen wir nicht, welches spezielle $h \in \mathcal{H}$ gleich oder am nächsten zu $\mathcal{C}$ ist. Sobald wir jedoch unsere Aufmerksamkeit auf diese Hypothesenklasse beschränken, reduziert sich das Erlernen einer Klasse auf das einfachere Problem, die vier Parameter zu finden, die h definieren.

Als Ziel gilt es, dasjenige $h \in \mathcal{H}$ zu finden, das so ähnlich wie möglich zu $\mathcal{C}$ ist. Nehmen wir an, die Hypothese h trifft eine Vorhersage für eine

Instanz $\boldsymbol{x}$, so dass gilt:

$$h(\boldsymbol{x}) = \begin{cases} 1 \text{ wenn } \boldsymbol{x} \text{ durch } h \text{ als positives Beispiel eingestuft wird,} \\ 0 \text{ wenn } \boldsymbol{x} \text{ durch } h \text{ als negatives Beispiel eingestuft wird.} \end{cases} \tag{2.5}$$

In der Realität kennen wir $\mathcal{C}(\boldsymbol{x})$ nicht, so dass wir nicht einschätzen können, wie gut $h(\boldsymbol{x})$ mit $\mathcal{C}(\boldsymbol{x})$ übereinstimmt. Wir haben lediglich einen Trainingsmenge $\mathcal{X}$, der nur eine kleine Teilmenge der Menge aller möglichen $\boldsymbol{x}$ darstellt. Der *empirische Fehler* misst den Anteil an Übungsinstanzen, bei denen die *Vorhersagen* von h nicht mit den *geforderten Werten* aus $\mathcal{X}$ übereinstimmen. Der Fehler der Hypothese h bei gegebenem Trainingsdatenssatz $\mathcal{X}$ ist

EMPIRISCHER FEHLER

$$E(h|\mathcal{X}) = \sum_{t=1}^{N} 1(h(\boldsymbol{x}^t) \neq r^t) \,. \tag{2.6}$$

Hierbei ist $1(a \neq b)$ gleich 1, wenn $a \neq b$, und gleich 0, wenn $a = b$ (siehe Abbildung 2.3).

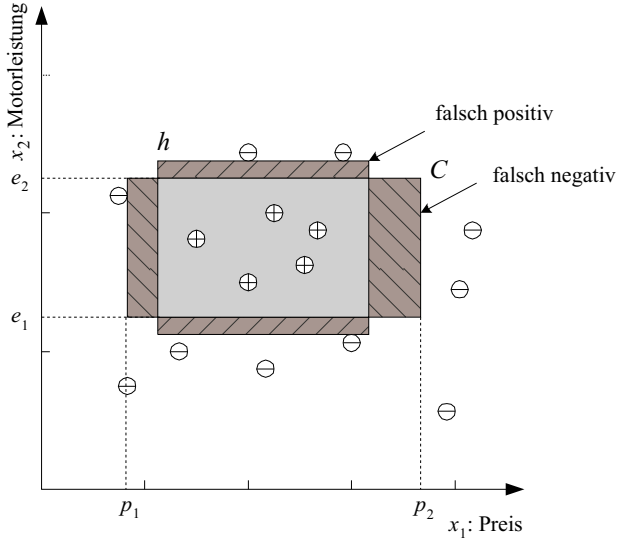


Abb. 2.3: C ist die eigentliche Klasse und h ist unsere induzierte Hypothese. Der Punkt, an dem C gleich 1, aber h gleich 0 ist, stellt eine falsch negative Zuordnung dar, und der Punkt, an dem C gleich 0, aber h gleich 1 ist, markiert eine falsch positive Zuordnung. Die anderen Punkte, also die richtig positiven und richtig negativen Zuordnungen, werden richtig klassifiziert.

In unserem Beispiel ist die Hypothesenklasse $\mathcal{H}$ die Menge aller möglichen Rechtecke. Jedes Quadrupel $(p_1^h, p_2^h, e_1^h, e_2^h)$ definiert eine Hypothese

h aus $\mathcal{H}$, und unsere Aufgabe liegt darin, die beste zu finden oder in anderen Worten, wir müssen bei gegebenem Übungssatz genau die vier Werte für die Parameter finden, die alle positiven Beispiele und keins der negativen Beispiele einschließen. Man beachte den Fall, wenn x_1 und x_2 reelle Zahlwerte annehmen; dann gibt es unendlich viele solcher h, auf die das zutrifft, für die also der Fehler E gleich null ist. Aber bei einem gegebenen zukünftigen Beispiel, welches irgendwo nahe den Grenzen zwischen positiven und negativen Beispielen liegt, könnten unterschiedliche in Frage kommende Hypothesen auch unterschiedliche Vorhersagen treffen. Hierbei handelt es sich um das Problem der *Generalisierung*, also um die Frage, wie oft unsere Hypothese zukünftige Beispiele, die nicht Teil der Trainingsmenge sind, korrekt einordnen wird.

GENERALISIERUNG

Eine Möglichkeit besteht darin, die *speziellste Hypothese* S zu finden, die dem engsten Rechteck entspricht, das alle positiven und keins der negativen Beispiele umschließt (siehe Abbildung 2.4). Damit erhalten wir eine Hypothese $h = S$ als unsere induzierte Klasse. Man beachte, dass die eigentliche Klasse $\mathcal{C}$ größer, aber niemals kleiner als S sein kann. Die *allgemeinste Hypothese* G ist das größte Rechteck, dass wir zeichnen können, das alle positiven und keins der negativen Beispiele enthält (Abbildung 2.4). Alle $h \in \mathcal{H}$ zwischen S und G sind gültige Hypothesen ohne Fehler, werden als *konsistent* zur Trainingsmenge angesehen und bilden zusammen den *Versionsraum*. Für eine andere gegebene Trainingsmenge, andere Hypothesen S, G und einen anderen Versionsraum können die Parameter, und somit die erlernte Hypothese h, anders aussehen.

SPEZIELLSTE HYPOTHESE

ALLGEMEINSTE HYPOTHESE

VERSIONSRAUM

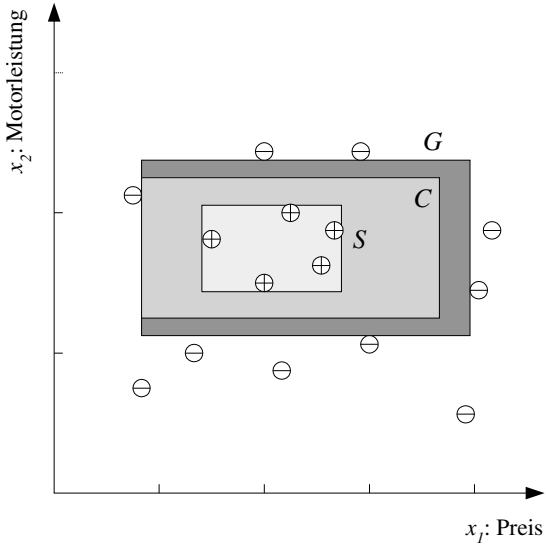


Abb. 2.4: S ist die speziellste Hypothese und G ist die allgemeinste Hypothese.

Tatsächlich wird es in Abhängigkeit von $\mathcal{X}$ und $\mathcal{H}$ verschiedene S_i und G_j geben, die jeweils die S-Menge und die G-Menge bilden. Jedes Element der S-Menge ist konsistent mit allen Instanzen, und es existieren keine Instanzen, die spezifischer sind. Entsprechend ist jedes Element der G-Menge konsistent mit allen Instanzen und es existiert keine konsistente Hypothese, die allgemeiner ist. Die beiden Mengen bilden zusammen die Grenzmenge, d. h., jede zwischen ihnen liegende Menge ist konsistent und Teil des Versionsraums. Es gibt einen als „Candidate Elimination" bezeichneten Algorithmus, der die S-Menge und die G-Menge inkrementell aktualisiert, wenn er nacheinander verschiedene Trainingsinstanzen vorgelegt bekommt (siehe Mitchell, 1997). Wir nehmen an, dass $\mathcal{X}$ hinreichend groß ist, damit S und G eindeutig sind.

Bei gegebenem $\mathcal{X}$ können wir S, G oder ein beliebiges h aus dem Versionsraum ableiten und als unsere Hypothese verwenden. Ein intuitiver Ansatz ist es, h in der Mitte zwischen S und G zu wählen. Das bedeutet, den *Rand* oder *Margin* zu vergrößern, womit der Abstand zwischen der Grenze und den ihr am nächsten liegenden Instanzen gemeint ist (Abbildung 2.5). Damit unsere Fehlerfunktion ein Minimum bei h mit maximalem Margin annimmt, sollten wir eine Fehlerfunktion (Verlustfunktion) verwenden, die nicht nur prüft, ob eine Instanz auf der richtigen Seite der Grenze liegt, sondern auch, wie weit sie von dieser entfernt ist. Das heißt, anstelle einer Funktion $h(\boldsymbol{x})$, die 0 oder 1 zurückgibt, brauchen wir eine, die uns ein Maß für den Abstand von der Grenze liefert. Außerdem brauchen wir eine Verlustfunktion, die diesen Wert verwendet, anstatt auf Gleichheit zu testen.

MARGIN

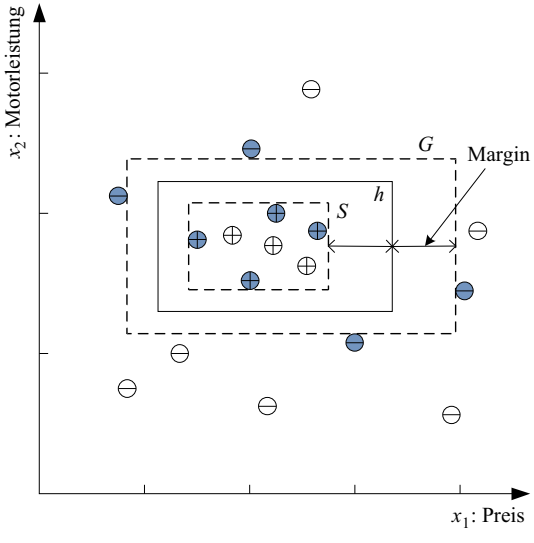


Abb. 2.5: Um die bestmögliche Separation zu erreichen, wählen wir die Hypothese mit dem größten Margin. Die ausgefüllten Instanzen sind diejenigen, die den Margin definieren (oder stützen); die anderen Instanzen können entfernt werden, ohne dass dies auf h Einfluss hat.

Bei manchen Anwendungen kann eine falsche Entscheidung sehr teuer sein. Dann können wir festlegen, dass jede Instanz, die zwischen S und G liegt, einen *Zweifelsfall* darstellt, den wir aufgrund unzureichender Daten nicht mit Sicherheit zuordnen können. Wenn ein solcher Fall auftritt, wird das System die Instanz *ablehnen* und die Entscheidung einem menschlichen Experten überlassen.

ZWEIFELSFALL

Hier nehmen wir an, dass $\mathcal{C}$ in $\mathcal{H}$ enthalten ist; das heißt, es existiert ein $h \in \mathcal{H}$, so dass $E(h|\mathcal{X})$ gleich null ist. Bei gegebener Hypothesenklasse $\mathcal{H}$ kann es durchaus passieren, dass wir $\mathcal{C}$ nicht erlernen können; das heißt, es existiert kein $h \in \mathcal{H}$, für das der Fehler gleich null ist. Somit müssen wir bei jeder Anwendung sicherstellen, dass $\mathcal{H}$ flexibel genug ist oder ausreichend „Kapazität“ hat, um $\mathcal{C}$ zu erlernen.

2.2 Vapnik-Chervonenkis(VC)-Dimension

Nehmen wir an, wir haben eine Datenmenge, der N Punkte beinhaltet. Diese N Punkte können auf 2^N verschiedene Weisen als positiv oder negativ eingeteilt werden. Daher können durch N Datenpunkte 2^N verschiedene Lernprobleme definiert werden. Wenn wir für jedes dieser Probleme eine Hypothese $h \in \mathcal{H}$ finden, welche die positiven von den negativen Beispielen trennt, dann sagen wir, $\mathcal{H}$ *zerlegt* N Punkte. Das heißt, jedes auf N Punkten definierte Lernproblem kann durch eine Hypothese aus $\mathcal{H}$ ohne Fehler gelernt werden. Die maximale Anzahl an Punkten, die durch $\mathcal{H}$ zerlegt werden können, nennt man die *Vapnik-Chervonenkis-Dimension* (VC-Dimension) von $\mathcal{H}$; sie wird mit $VC(\mathcal{H})$ bezeichnet und misst die Mächtigkeit der Hypothesenklasse $\mathcal{H}$.

VAPNIK-CHERVONENKIS-DIMENSION

Aus Abbildung 2.6 erkennen wir, dass ein axial ausgerichtetes Rechteck vier Punkte in zwei Dimensionen zerlegen kann. In diesem Fall ist $VC(\mathcal{H})$ gleich vier, wenn $\mathcal{H}$ die Hypothesenklasse von axial ausgerichteten Rechtecken in zwei Dimensionen ist. Bei der Berechnung der VC-Dimension reicht es aus, dass wir vier Punkte finden, die zerlegt werden können; es ist nicht notwendig, dass wir in der Lage sind, *beliebige* vier Punkte in zwei Dimensionen zu zerlegen.

Zum Beispiel können vier auf einer Linie platzierte Punkte nicht durch Rechtecke zerlegt werden. Allerdings können wir fünf Punkte *beliebig* in zwei Dimensionen verteilen, so dass ein Rechteck die positiven und negativen Beispiele für alle möglichen Einteilungen trennen kann.

Die VC-Dimension mag pessimistisch erscheinen. Sie sagt uns, dass wir bei Benutzung eines Rechtecks als Hypothesenklasse nur die Datensätze erlernen können, die vier Punkte enthalten, nicht mehr. Ein Lernalgorithmus, der Datensätze von vier Punkten erlernen kann, ist nicht sehr nützlich. Das liegt jedoch daran, dass die VC-Dimension unabhängig

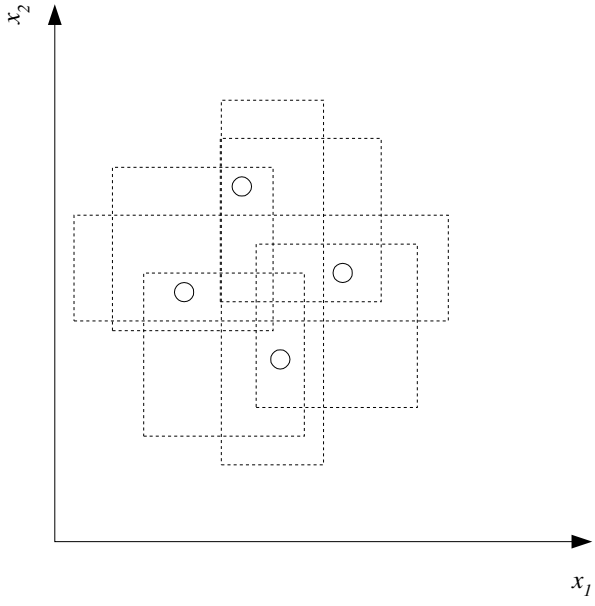


***Abb. 2.6*:** Ein axial ausgerichtetes Rechteck kann 4 Punkte zerlegen. Nur Rechtecke, die zwei Punkte abdecken, sind dargestellt.

von der Wahrscheinlichkeitsverteilung ist, aus der die Instanzen gezogen werden. In der Realität verändert sich die Welt fließend, nahe beieinander liegende Instanzen sind zumeist dem gleichen Typ zugeordnet und wir brauchen uns nicht um *alle möglichen Einteilungen* der Instanzen zu sorgen. Es gibt eine Vielzahl von Datensätzen, die viel mehr als vier Datenpunkte enthalten und die durch unsere Hypothesenklasse erlernbar sind (Abbildung 2.1). Daher sind also auch Hypothesenklassen mit kleiner VC-Dimension anwendbar und denen mit großer VC-Dimension vorzuziehen, wie zum Beispiel einer Lookup-Tabelle mit unendlicher VC-Dimension.

2.3 PAC-Lernen

Unter Benutzung des engsten Rechtecks S als unsere Hypothese möchten wir herausfinden, wie viele Beispiele wir benötigen. Wir erhoffen uns von unserer Hypothese, dass sie annähernd korrekt ist, sprich, dass die Fehlerwahrscheinlichkeit durch gewisse Werte eingegrenzt ist. Wir möchten unserer Hypothese auch insofern vertrauen, als dass wir sicher sein wollen, dass unsere Hypothese in den meisten Fällen (wenn nicht in allen) korrekt sein wird; also wollen wir auch wahrscheinlich richtig liegen (mit einer von uns festgelegten Wahrscheinlichkeit).

Beim *Probably Approximately Correct Learning* (PAC-Lernen; dt.: wahrscheinlich annähernd richtiges Lernen) wollen wir bei gegebener Klasse $\mathcal{C}$ und Beispielen, die nach einer unbekannten aber festen Wahrscheinlichkeitsverteilung $p(x)$ gezogen werden, die Anzahl an Beispielen N herausfinden, durch welche die Hypothese h mit einer Wahrscheinlichkeit von mindestens $1 - \delta$ einen Fehler von höchstens ϵ aufweist, und zwar für beliebige $\delta \leq 1/2$ und $\epsilon > 0$,

PAC-Lernen

$$P\{\mathcal{C}\Delta h \leq \epsilon\} \geq 1 - \delta\,,$$

wobei $\mathcal{C}\Delta h$ die Differenzregion zwischen $\mathcal{C}$ und h bezeichnet.

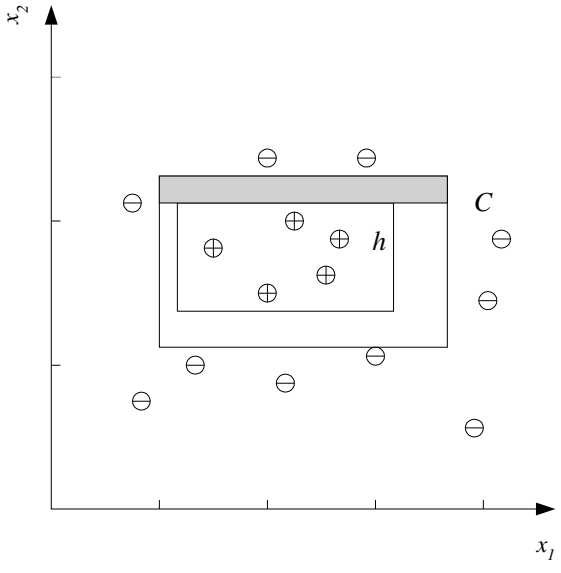


Abb. 2.7: Die Differenz zwischen h und $\mathcal{C}$ ist die Summe aus vier rechteckigen Streifen, von denen einer schattiert dargestellt ist.

Da S das am engsten gefasste Rechteck ist, ergibt sich in unserem Fall die Fehlerregion zwischen $\mathcal{C}$ und $h = S$ aus der Summe von vier rechteckigen Streifen (siehe Abbildung 2.7). Wir möchten sicherstellen, dass die Wahrscheinlichkeit dafür, dass ein positives Beispiel in diesen Bereich fällt (und einen Fehler verursacht) höchstens ϵ beträgt. Wenn wir für jeden dieser Streifen garantieren können, dass für diesen Streifen diese Wahrscheinlichkeit nach oben durch $\epsilon/4$ beschränkt ist, dann tritt der Gesamtfehler mit einer Wahrscheinlichkeit von höchstens $4\epsilon/4 = \epsilon$ ein. Man beachte, dass wir die Überlappungen in den Ecken zweimal zählen, und dass der gesamte tatsächliche Fehler in diesem Fall weniger als $4(\epsilon/4)$ beträgt. Die Wahrscheinlichkeit dafür, dass ein zufällig gezogenes Beispiel diesen Streifen nicht trifft, liegt bei $1 - \epsilon/4$. Die Wahrscheinlichkeit, dass alle N unabhängigen Ziehungen den Streifen verfehlen, beträgt $(1-\epsilon/4)^N$,

und mit einer Wahrscheinlichkeit von höchstens $4(1-\epsilon/4)^N$ verfehlen alle N unabhängigen Ziehungen irgendeinen der vier Streifen; dieser Wert soll nach Möglichkeit maximal δ sein. Wir erhalten die Ungleichung

$$(1-x) \leq \exp[-x] \,.$$

Wenn wir also N und δ so wählen, dass wir

$$4\exp[-\epsilon N/4] \leq \delta$$

erhalten, dann können wir ebenfalls $4(1-\epsilon/4)^N \leq \delta$ schreiben. Wenn wir beide Seiten durch 4 teilen, den (natürlichen) Logarithmus ziehen und die Terme umformen, erhalten wir

$$N \geq (4/\epsilon)\log(4/\delta) \,. \tag{2.7}$$

Unter der Vorraussetzung, dass wir mindestens $(4/\epsilon)\log(4/\delta)$ unabhängige Beispiele aus $\mathcal{C}$ ziehen und das engstmögliche Rechteck als unsere Hypothese h wählen, wird mit der *Konfidenzwahrscheinlichkeit* von mindestens $1-\delta$ ein gegebener Punkt mit einer *Fehlerwahrscheinlichkeit* von maximal ϵ falsch klassifiziert. Wir können die Konfidenz auf einen beliebig großen Wert setzen, indem wir δ verringern; durch Verringerung von ϵ erzielen wir einen beliebig kleinen Fehler. In Gleichung 2.7 erkennen wir, dass die Anzahl an Beispielen eine jeweils linear bzw. logarithmisch langsam wachsende Funktion von $1/\epsilon$ und $1/\delta$ ist.

2.4 Rauschen

Rauschen

Mit *Rauschen* (engl. *noise*) bezeichnet man jede ungewollte Anomalie in Daten; bedingt durch Rauschen kann eine Klasse schwerer zu erlernen sein, und ein Fehler gleich null ist mit einer einfachenHypothesenklasse möglicherweise nicht zu erzielen (siehe Abbildung 2.8). Es existieren diverse Interpretationen für das Rauschen:

- Eine Ursache kann Ungenauigkeit bei der Aufzeichnung der Eingabeattribute sein, wodurch sich die Datenpunkte im Eingaberaum verschieben.
- Es könnte Fehler in den Einteilungen der Datenpunkte geben, durch die positive Instanzen als negative ausgewiesen werden und umgekehrt. Dies nennt man manchmal *Lehrerrauschen* (engl. *teacher noise*).
- Es können zusätzliche Attribute vorhanden sein, die wir nicht in unsere Betrachtungen einbezogen haben, die aber die Einstufung einer Instanz als positiv bzw. negativ beeinflussen. Derlei Attribute

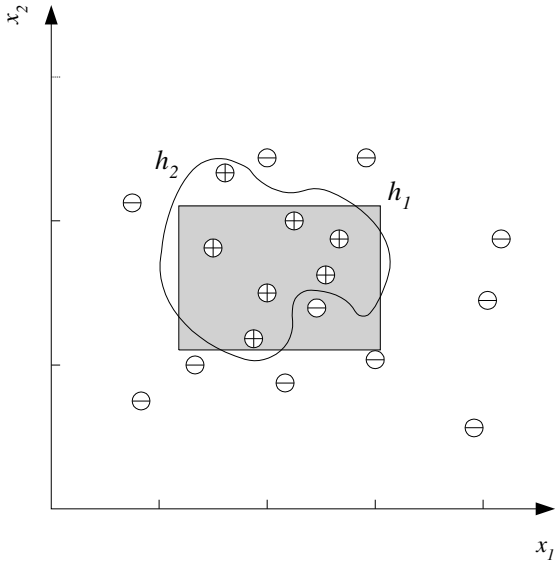


***Abb. 2.8*:** Bei Vorhandensein von Rauschen lassen sich die positiven und negativen Instanzen nicht so leicht voneinander abgrenzen. Die vollständige Vermeidung von Fehlklassifizierungen liegt unter Umständen mit einer einfachen Hypothese nicht im Bereich des Möglichen. Ein Rechteck ist eine einfache Hypothese mit vier Parametern, welche die Ecken definieren. Eine willkürliche geschlossene Form kann mittels Teilfunktionen mit einer größeren Zahl von Kontrollpunkten gezeichnet werden.

können *verborgen* oder *latent*, also nicht beobachtbar sein. Der Effekt solcher vernachlässigten Attribute wird somit als zufällige Komponente modelliert und ist im „Rauschen“ enthalten.

Wie aus Abbildung 2.8 ersichtlich wird, gibt es bei Vorhandensein von Rauschen keine einfache Grenze zwischen positiven und negativen Instanzen und sie zu trennen erfordert eine kompliziertere Hypothese, die einer Hypothesenklasse von größerer Mächtigkeit entspricht. Während ein Rechteck durch vier Werte definiert werden kann, erfordert die Definition einer komplizierteren Form ein komplexeres Modell mit einer viel größeren Zahl an Parametern. Ein komplexes Modell lässt sich perfekt auf die Daten anpassen und ein Fehler gleich null liegt im Rahmen des Möglichen; die geschlängelte Form aus Abbildung 2.8 verdeutlicht dies. Eine weitere Möglichkeit besteht darin, das Modell einfach zu halten und einen gewissen Fehlergrad zu erlauben; man betrachte hierzu das Rechteck in Abbildung 2.8.

Die Benutzung des einfachen Rechtecks erscheint sinnvoller (es sei denn, der zugehörige Übungsfehler ist viel größer), denn:

1. Es ist ein leicht zu benutzendes Modell. Es ist einfach zu erkennen, ob ein Punkt innerhalb oder außerhalb des Rechtecks liegt, und wir können für zukünftige Dateninstanzen leicht prüfen, ob es sich um positive oder negative Instanzen handelt.

2. Das Modell kann leicht trainiert werden und besitzt weniger Parameter. Es ist einfacher, die Eckwerte eines Rechtecks als die Kontrollpunkte einer willkürlichen Form zu finden. Mit einem kleinen Übungssatz erwarten wir vom einfacheren Modell bei leichten Unterschieden in den Trainingsinstanzen geringere Veränderungen als beim komplexen Modell. Bei einem einfachen Modell spricht man daher von geringerer *Varianz*. Andererseits basiert ein einfaches Modell mehr auf Vereinfachung der Realität, ist starrer und kann versagen, wenn die zugrundeliegende Klasse tatsächlich nicht so simpel ist, denn ein einfacheres Modell unterliegt einer größeren *Verzerrung* (bias). Die Suche nach dem optimalen Modell entspricht der Minimierung sowohl des systematischen Fehlers als auch der Varianz.

3. Es ist ein leicht zu erklärendes Modell. Ein Rechteck entspricht einfach der Definition von Intervallen für zwei Attribute. Durch das Erlernen eines einfachen Modells können wir Informationen aus den in der Datenmenge gegebenen Rohdaten extrahieren.

4. Wenn es tatsächlich fälschlich eingeteilte Instanzen oder Rauschen in den Eingabewerten gibt, und wenn die tatsächliche Klasse wirklich ein simples Modell wie das Rechteck ist, dann wird das Rechteck aufgrund seiner geringeren Varianz und niedrigeren Sensititvität gegenüber einzelnen Instanzen ein besserer Diskriminator sein als die geschlängelte Form, obwohl die einfache Form mehr Fehler auf der Trainingsmenge aufweisen mag. Wir sprechen davon, dass ein einfaches (aber nicht zu einfaches) Modell besser generalisieren würde als ein komplexes Modell. Dieses Prinzip ist bekannt als *Ockhams Rasiermesser* und fordert unter Verweis auf die *höhere Plausibilität von einfacheren Erklärungen*, dass jegliche unnötige Komplexität „wegrasiert“ werden möge.

Ockhams Rasiermesser

2.5 Erlernen multipler Klassen

In unserem Beispiel des Erlernens der Klasse Familienwagen haben wir positive Beispiele, die zur Klasse Familienwagen gehören und negative Beispiele, die zu beliebigen anderen Autotypen gehören. Dies ist ein *Zweiklassenproblem*. Im allgemeinen Fall haben wir K Klassen, geschrieben als $\mathcal{C}_i, i = 1, \ldots, K$, und eine Eingabeinstanz gehört zu genau einer dieser Klassen. Die Trainingsmenge ist nun von der Form

$$\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N ,$$

wobei $\boldsymbol{r}$ hier K Dimensionen besitzt und

$$r_i^t = \begin{cases} 1 \text{ falls } \boldsymbol{x}^t \in \mathcal{C}_i \,, \\ 0 \text{ falls } \boldsymbol{x}^t \in \mathcal{C}_j, j \neq i \,. \end{cases} \tag{2.8}$$

In Abbildung 2.9 ist ein Beispiel mit Instanzen aus drei Klassen gegeben: Familienwagen, Sportwagen und Luxuslimousine.

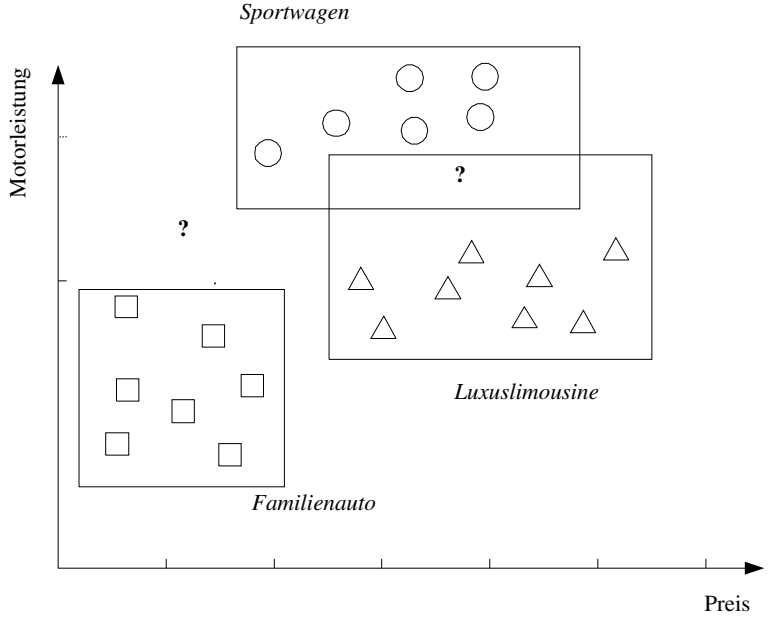


***Abb. 2.9*:** Es gibt drei Klassen: Familienwagen, Sportwagen und Luxuslimousine. Drei Hypothesen werden induziert, wobei jede die Instanzen einer der Klassen abdeckt, aber die Instanzen der anderen beiden Klassen außen vor lässt. „?“ repräsentiert abgelehnte Regionen, in denen keine oder mehr als eine Klasse gewählt wurde.

Beim maschinellen Lernen zu Klassifikationszwecken möchten wir die Grenze bestimmen, durch welche die Instanzen einer Klasse von denen aller anderen Klassen getrennt werden. Somit betrachten wir ein Klassifikationsproblem mit K Klassen als K Zweiklassenprobleme. Die Übungsbeispiele, die zu $\mathcal{C}_i$ gehören, sind die positiven Instanzen von Hypothese h_i, und die Beispiele aller anderen Klassen stellen die negativen Instanzen von h_i dar. In einem K-Klassenproblem haben wir also K Hypothesen zu lernen, so dass gilt:

$$h_i(\boldsymbol{x}^t) = \begin{cases} 1 \text{ falls } \boldsymbol{x}^t \in C_i \,, \\ 0 \text{ falls } \boldsymbol{x}^t \in \mathcal{C}_j, j \neq i \,. \end{cases} \tag{2.9}$$

Der gesamte empirische Fehler ist die Summe über die Vorhersagen für alle Klassen über alle Instanzen:

$$E\left(\{h_i\}_{t=1}^{K}|\mathcal{X}\right) = \sum_{t=1}^{N}\sum_{i=1}^{K} 1\left(h_i(\boldsymbol{x}^t) \neq r_i^t\right) \ . \tag{2.10}$$

Für ein gegebenes $\boldsymbol{x}$ ist im Idealfall nur eines der $h_i(\boldsymbol{x}), i = 1, \ldots, K$ gleich eins und wir können eine Klasse wählen. Wenn aber keines oder zwei oder mehrere $h_i(\boldsymbol{x})$ gleich eins sind, können wir keine Klasse wählen und sprechen hier von einem *Zweifelsfall*, der vom Klassifikator *abgelehnt* wird.

Ablehnung

In unserem Beispiel des Erlernens der Klasse Familienwagen haben wir nur eine Hypothese benutzt und nur die positiven Beispiele modelliert. Jegliche negativen Beispiele außerhalb sind keine Familienwagen. Als Alternative ziehen wir es möglicherweise manchmal vor, zwei Hypothesen zu erstellen, eine für die positiven und die andere für die negativen Instanzen. Dies setzt voraus, dass auch die negativen Instanzen eine Struktur haben, die von einer anderen Hypothese abgedeckt werden können. Die Unterscheidung zwischen Familienwagen und Sportwagen ist solch ein Problem; jede Klasse hat ihre eigene Struktur. Der Vorteil liegt darin, dass im Falle einer Luxuslimousine als Eingabe beide Hypothesen eine negative Entscheidung treffen und die Eingabe abweisen können.

Wenn wir erwarten, in einer Datenbasis alle Klassen mit ähnlichen Verteilungen – Formen im Eingaberaum – zu haben, dann können wir die gleiche Hypothesenklasse für alle Klassen benutzen. Bei einer Datenbasis für die Erkennung handgeschriebener Ziffern erwarten wir, dass alle Ziffern ähnliche Verteilungen haben. Dagegen haben wir in einer Datenbasis, die in der medizinischen Diagnostik eingesetzt wird, zwei Klassen von Personen – gesunde und kranke –, so dass wir mit völlig verschiedenen Verteilungen für die beiden Gruppen rechnen müssen. Dass sich das Kranksein auf verschiedene Weisen äußern kann, spiegelt sich in den Eingaben wider: Alle gesunden Personen ähneln sich in dieser Hinsicht, während jede kranke Person auf ihre eigene Weise krank ist.

2.6 Regression

Bei der Klassifikation sind die für eine gegebene Eingabe generierten Ausgabewerte vom Typ Boolean; es handelt sich um ja/nein-Antworten. Wenn die Ausgabewerte numerisch sind, wollen wir keine Klasse der Form $\mathcal{C}(\boldsymbol{x}) \in \{0, 1\}$ erlernen, sondern eine kontinuierliche Funktion. Beim maschinellen Lernen ist die Funktion zwar nicht bekannt, jedoch haben wir eine Trainingsmenge von Beispielen, die aus ihr gezogen wurden:

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^{N} \ ,$$

wobei $r^t \in \mathbb{R}$. Wenn kein Rauschen vorhanden ist, besteht die Aufgabe in der *Interpolation*. Wir wollen die Funktion $f(x)$ finden, die durch diese Punkte geht, so dass wir

INTERPOLATION

$$r^t = f(\boldsymbol{x}^t)$$

erhalten.

Bei der *Polynominterpolation* finden wir für N gegebene Punkte das Polynom vom $(N-1)$-ten Grad, welches wir nutzen können, um die Ausgabewerte für beliebige $\boldsymbol{x}$ vorherzusagen. Dies nennt man *Extrapolation*, wenn x außerhalb des Bereichs der Trainigsbeispiele $\boldsymbol{x}^t$ liegt. Bei Zeitreihenvorhersagen beispielsweise haben wir Daten bis zur Gegenwart und wollen den Wert für die Zukunft vorhersagen. Bei der *Regression* sind die Ausgaben der unbekannten Funktion verrauscht

EXTRAPOLATION

REGRESSION

$$r^t = f(\boldsymbol{x}^t) + \epsilon\,, \tag{2.11}$$

wobei $f(\boldsymbol{x}) \in \mathbb{R}$ die unbekannte Funktion darstellt und ϵ für zufälliges Rauschen steht. Die Erklärung für das Rauschen liegt darin, dass es zusätzliche *verborgene* Variablen gibt, die sich unserer Beobachtung entziehen:

$$r^t = f^*(\boldsymbol{x}^t, \boldsymbol{z}^t)\,, \tag{2.12}$$

wobei $\boldsymbol{z}^t$ diese verborgenen Variablen repräsentiert. Wir wollen gerne die Ausgabe mit Hilfe unseres Modells $g(\boldsymbol{x})$ approximieren. Der empirische Fehler für die Trainingsmenge $\mathcal{X}$ ist

$$E(g|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - g(\boldsymbol{x}^t)]^2\,. \tag{2.13}$$

Da r und $g(\boldsymbol{x})$ numerisch sind, beispielsweise $\in \mathbb{R}$, gibt es eine definierte Ordnung für ihre Werte. Somit können wir eine *Distanz* zwischen Werten als das Quadrat der Differenz definieren, durch die wir mehr Informationen erhalten, als uns die Aussage gleich/ungleich liefern würde, wie dies bei der Klassifikation üblich ist. Das Quadrat der Differenz ist nur eine mögliche Fehlerfunktion; eine andere findet sich im Absolutwert der Differenz. In den folgenden Kapiteln werden wir auf weitere Beispiele treffen.

Unser Ziel ist es, das $g(\cdot)$ zu finden, welches den empirischen Fehler minimiert. Unser Vorgehen ist dabei wieder dasselbe. Wir gehen also von einer Hypothesenklasse mit einer geringen Anzahl an Parametern für $g(\cdot)$ aus. Wenn wir annehmen, dass $g(\boldsymbol{x})$ linear ist, ergibt sich:

$$g(\boldsymbol{x}) = w_1 x_1 + \cdots + w_d x_d + w_0 = \sum_{j=1}^{d} w_j x_j + w_0\,. \tag{2.14}$$

Kehren wir nun zurück zu unserem Beispiel aus Abschnitt 1.2.3, in dem wir den Preis eines Gebrauchtwagens geschätzt haben. Dort haben wir ein lineares Modell der folgenden Form mit einem einzigen Eingabewert benutzt:

$$g(x) = w_1 x + w_0 \,, \tag{2.15}$$

wobei w_1 und w_0 die anhand der Daten zu erlernenden Parameter sind. Die Werte für w_1 und w_0 sollten die folgende Gleichung minimieren:

$$E(w_1, w_0|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}[r^t - (w_1 x^t + w_0)]^2 \,. \tag{2.16}$$

Das Minimum kann hierfür berechnet werden, indem man die partiellen Ableitungen von E hinsichtlich w_1 und w_0 nimmt, sie gleich 0 setzt und sie für die zwei Unbekannten löst:

$$\begin{aligned} w_1 &= \frac{\sum_t x^t r^t - \overline{x}\overline{r}N}{\sum_t (x^t)^2 - N\overline{x}^2} \\ w_0 &= \overline{r} - w_1 \overline{x} \end{aligned} \tag{2.17}$$

mit $\overline{x} = \sum_t x^t/N$, $\overline{r} = \sum_t r^t/N$. Die dadurch gefundene Gerade ist in Abbildung 1.2 dargestellt.

Ist das lineare Modell zu einfach, unterliegt es zu vielen Einschränkungen und verursacht einen hohen Näherungsfehler; in solch einem Fall kann die Ausgabe als Funktion höherer Ordnung der Eingabe angenommen werden, beispielsweise als quadratische Funktion:

$$g(x) = w_2 x^2 + w_1 x + w_0 \,. \tag{2.18}$$

Indem wir genauso vorgehen wie im linearen Fall, erhalten wir analytische Lösungen für die Parameter. Wird der Grad der Polynome erhöht, verringert sich der Fehler auf den Trainigsdaten. Allerdings folgt ein Polynom höheren Grades einzelnen Punkten sehr genau, anstatt einen allgemeinen Trend zu erfassen (man betrachte hier das Polynom sechsten Grades aus Abbildung 2.10). Wir müssen also vorsichtig sein, wenn wir die Komplexität des Modells an die Komplexität der den Daten zugrundeliegenden Funktion feinanpassen.

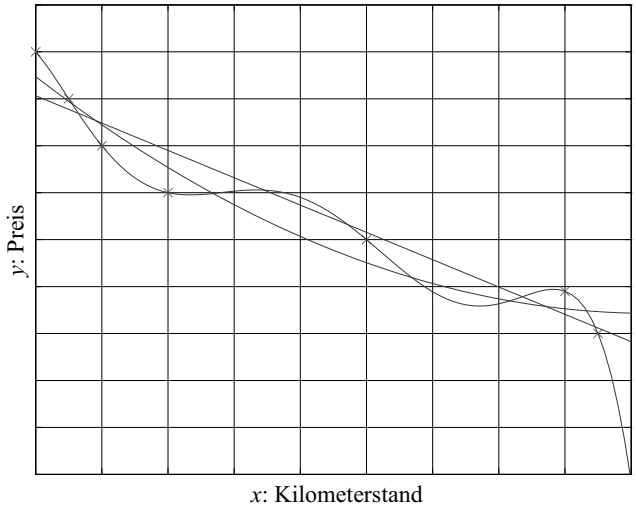


***Abb. 2.10*:** Lineare Polynome zweiten und sechsten Grades werden an denselben Satz von Punkten angepasst. Der höhere Grad sorgt für eine perfekte Anpassung, aber in Anbetracht der Datenmenge ist es sehr unwahrscheinlich, dass die reale Kurve eine derartige Form annimmt. Der zweite Grad erscheint besser als die lineare Anpassung, was das Erfassen des Trends in den Übungsdaten angeht.

2.7 Modellauswahl und Generalisierung

Beginnen wir mit einem Fall, in dem eine Boolesche Funktion anhand von Beispielen erlernt werden soll. Bei einer Booleschen Funktion sind alle Eingabe- und Ausgabewerte binär. Es gibt 2^d Möglichkeiten, d binäre Werte zu belegen, und somit hat die Trainingsmenge bei d Eingabewerten höchstens 2^d Beispiele. Wie in Tabelle 2.1 dargestellt, kann jedes dieser mit der Markierung 0 oder 1 versehen werden, wodurch sich 2^{2^d} mögliche Boolesche Funktionen für d Eingabewerte ergeben.

***Tabelle 2.1*:** Für zwei Eingabewerte gibt es vier mögliche Fälle und sechzehn mögliche Boolesche Funktionen.

x_1	x_2	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	h_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Jedes einzelne Trainingsbeispiel eliminiert die Hälfte aller Hypothesen, nämlich die, deren Annahmen falsch sind. Nehmen wir zum Beispiel an,

wir haben $x_1 = 0$, $x_2 = 1$ und der Ausgabewert ist 0; dadurch werden $h_5, h_6, h_7, h_8, h_{13}, h_{14}, h_{15}, h_{16}$ eliminiert. Das ist eine Möglichkeit, den Lernprozess zu betrachten. Während wir weitere Beispiele erhalten, entfernen wir all die Hypothesen, die mit den Daten nicht konsistent sind. Um nur eine Hypothese übrig zu behalten, müssen wir im Falle einer Booleschen Funktion *alle* 2^d Beispiele betrachten. Wenn der uns vorliegende Übungssatz nur eine kleine Teilmenge aller möglichen Instanzen enthält, so wie das zumeist der Fall ist – sprich, wenn wir wissen wie die Ausgabewerte für lediglich einen kleinen Prozentsatz aller Fälle aussehen – so ist die Lösung nicht eindeutig. Nachdem wir N Beispielfälle gesehen haben, bleiben 2^{2^d-N} mögliche Funktionen übrig. Dies ist ein Beispiel für ein *unbestimmtes Problem*, bei dem die Daten an sich nicht ausreichen, um eine eindeutige Lösung zu finden.

UNBESTIMMTES PROBLEM

Das gleiche Problem existiert auch bei anderen Lernanwendungen, bei der Klassifikation und bei der Regression. Je mehr Übungsbeispiele wir sehen, desto mehr wissen wir auch über die zugrundeliegende Funktion und desto mehr inkonsistente Hypothesen können wir aus der Hypothesenklasse eliminieren; wir bleiben jedoch immer noch auf einer ganzen Menge konsistenter Hypothesen sitzen.

Weil nun das Lernen eine unbestimmte Ausgangsbasis besitzt und die Daten an sich nicht ausreichen, um eine Lösung zu finden, sollten wir zusätzliche Annahmen treffen, um für die uns vorliegenden Daten eine eindeutige Lösung zu erhalten. Die Menge an Annahmen, die wir treffen, um das Lernen zu ermöglichen, nennt man die *induktive Verzerrung* des Lernalgorithmus. Eine Ursache für die induktive Verzerrung ist die Annahme einer Hypothesenklasse $\mathcal{H}$. Beim Lernen der Klasse Familienwagen gibt es unendlich viele Wege, die positiven von den negativen Beispielen zu trennen. Die Vermutung der Form eines Rechtecks stellt eine Art der induktiven Verzerrung dar. Das Rechteck mit dem kleinsten Margin ist eine weitere Art der induktiven Verzerrung. Bei der linearen Regression liefert die Annahme einer linearen Funktion eine induktive Verzerrung; wird unter allen Linien diejenige gewählt, die den quadratischen Fehler minimiert, ergibt sich eine andere induktive Verzerrung.

INDUKTIVE VERZERRUNG

Wir wissen aber, dass jede Hypothesenklasse von gewisser Mächtigkeit ist und nur bestimmte Funktionen erlernen kann. Die erlernbare Klasse von Funktionen kann erweitert werden, indem man eine Hypothesenklasse von größerer Mächtigkeit nutzt, die komplexere Hypothesen enthält. Beispielsweise weist die Hypothesenklasse, die sich aus der Vereinigung von zwei nichtüberlappenden Rechtecken ergibt, eine höhere Mächtigkeit auf, jedoch sind ihre Hypothesen auch komplexer. Ähnliches trifft auf die Regression zu; wenn wir den Grad der Polynome erhöhen, nehmen auch Mächtigkeit und Komplexität zu. Die Frage ist daher, an welchem Punkt man aufhören sollte.

Das Lernen ist also ohne induktive Verzerrung nicht möglich, und nun stellt sich die Frage, wie man den richtigen Grad an Verzerrung findet.

Dies bezeichnet man als *Modellselektion*. Bei der Beantwortung dieser Frage sollten wir bedenken, dass das Ziel beim maschinellen Lernen kaum darin besteht, Daten zu replizieren, sondern Vorhersagen für neue Daten zu treffen. Das heißt, wir wären gerne in der Lage, die richtigen Ausgabewerte für eine Eingabeinstanz außerhalb des Satzes zu generieren, also für eine, deren korrekte Ausgabewerte nicht in der Trainingsmenge vorgegeben sind. Die Qualität, mit der ein mit Datensätzen trainiertes Modell die richtigen Ausgaben für neue Instanzen vorhersagt, nennt man *Generalisierung*.

MODELLSELEKTION

GENERALISIERUNG

Für die bestmögliche Generalisierung müssten wir die Komplexität der Hypothese mit der den Daten zugrundeliegenden Funktion abgleichen. Wenn die Hypothese weniger komplex ist als die Funktion, sprechen wir von *Unteranpassung* (underfitting). Ein Beispiel hierfür ist der Versuch, eine Gerade auf eine Datenstichprobe anzupassen, die von einem Polynom dritten Grades stammt. Wenn wir in solch einem Fall die Komplexität erhöhen, nehmen sowohl der Trainingsfehler als auch der Validierungsfehler ab. Wenn wir aber eine Hypothese $\mathcal{H}$ haben, die zu komplex ist, reichen die Daten nicht aus, um sie zu vereinfachen und das Ergebnis ist letztendlich möglicherweise eine unnütze Hypothese $h \in \mathcal{H}$, zum Beispiel wenn zwei Rechtecke auf Daten angepasst werden, die aus nur einem Rechteck entnommen wurden. Oder aber beim Vorhandensein von Rauschen wird eine zu komplexe Hypothese möglicherweise nicht nur die zugrundeliegende Funktion erlernen, sondern auch das Rauschen in den Daten. Die Folge ist eine schlechte Anpassung, beispielsweise wenn ein Polynom sechsten Grades auf verrauschte Daten, die mit einem Polynom dritten Grades erhoben wurden, angepasst wird. In dem Fall spricht man von *Überanpassung* (overfitting). Hierbei hilft das Vorhandensein von mehr Übungsdaten unter Umständen, jedoch auch nur bis zu einem gewissen Punkt. Zu einer gegebenen Trainingsmenge und einem gegebenem $\mathcal{H}$ können wir das $h \in \mathcal{H}$ mit dem minimalen Trainingsfehler bestimmen, doch wenn $\mathcal{H}$ nicht gut gewählt ist, dann werden wir unabhängig davon, welches $h \in \mathcal{H}$ wir nehmen, keine gute Generalisierung bekommen.

UNTERANPASSUNG

ÜBERANPASSUNG

Wir können unsere Betrachtungen unter Verweis auf den *dreifachen Kompromiss* (triple trade-off) (Dietterich 2003) zusammenfassen. Bei allen Lernalgorithmen, die mit Beispieldaten trainiert werden, gibt es einen Kompromiss zwischen den folgenden drei Faktoren:

DREIFACHER KOMPROMISS

- die Komplexität der Hypothese, die wir auf die Daten anpassen, also die Mächtigkeit der Hypothesenklasse,
- die Menge an Daten und
- der Generalisierungsfehler bei neuen Beispielen.

Mit zunehmender Menge an Daten nimmt der Generalisierungsfehler ab. Wenn die Komplexität der Modellklasse $\mathcal{H}$ sich erhöht, verringert sich

der Generalisierungsfehler zunächst, nimmt aber dann wieder zu. Der Generalisierungsfehler einer zu komplexen Hypothese $\mathcal{H}$ kann im Rahmen gehalten werden, indem die Menge an Übungsdaten erhöht wird, dies aber auch wieder nur bis zu einem gewissen Punkt. Wenn die Daten von einer Geraden genommen wurden und wir an ein Polynom höherer Ordnung anpassen wollen, dann wird die angepasste Kurve zwangsläufig nahe an der Gerade liegen, wenn es Trainingsdaten in der Umgebung gibt. Wo es es keine Trainingsdaten gibt, wird sich ein Polynom höherer Ordnung eventuell erratisch verhalten.

Wir können die Fähigkeit zur Generalisierung einer Hypothese messen, also die Qualität ihrer induktiven Verzerrung, wenn wir Zugriff auf Daten außerhalb der Trainingsmenge haben. Wir simulieren dies, indem wir die uns vorliegenden Trainingsmenge in zwei Teile zerlegen. Den einen Teil nutzen wir zum Trainieren (d. h. um eine Hypothese zu finden) und den verbleibenden Teil, genannt *Validierungsmenge*, nutzen wir, um die Fähigkeit zur Generalisierung zu testen. Das heißt, für eine gegebene Menge möglicher Hypotheseklassen $\mathcal{H}_i$ passen wir jeweils das beste $h_i \in \mathcal{H}_i$ an die Trainingsmenge an. Unter der Annahme, dass Trainings- und Validierungsdmenge groß genug sind, ist die Hypothese mit größter Genauigkeit hinsichtlich der Validierungsmenge die beste (diejenige mit der besten induktiven Verzerrung). Diesen Vorgang nennt man *Kreuzvalidierung*. Um also beispielsweise bei der polynomialen Regression den richtigen Grad für eine gegebene Menge an zur Auswahl stehenden Polynomen verschiedenen Grades zu finden, wobei Polynome verschiedener Grade den verschiedenen $\mathcal{H}_i$ entsprechen, nehmen wir zunächst ihre Koeffizienten für die Trainingsmenge, berechnen ihre Fehler für den Validierungsmenge, und wählen dann das Polynom mit dem geringsten Validierungsfehler als das beste Polynom aus.

VALIDIERUNGSMENGE

KREUZVALIDIERUNG

Manchmal ist es nötig, den Fehler explizit auszuweisen, um einen Eindruck von der Größenordnung des erwarteten Fehlers unseres besten Modells zu vermitteln. In dem Fall müssen wir darauf achten, dass wir nicht den Validierungsfehler angeben. Wir haben die Validierungsmenge genutzt, um unser bestes Modell zu finden, und im Endeffekt wurde diese damit ein Teil der Trainingsmenge. Wir benötigen eine dritten Menge, einen *Testmenge*, manchmal auch *Publikationsmenge* genannt, welche Beispiele enthält, die weder beim Trainieren noch bei der Validierung genutzt wurden. Eine Analogie aus unserem Leben findet sich im Beispiel einer Universitätsvorlesung: die ausgewählten Probleme, welche die Lehrkraft während des Unterrichts löst, bilden die Trainingsmenge; Prüfungsfragen stellen die Validierungsmenge dar; und die Problemstellungen, die uns später in der Praxis begegnen, bilden die Testmenge.

TESTMENGE

Die Unterteilung in Trainingsdaten und Validierungsdaten können wir nicht dauerhaft beibehalten, da die Validierungsdaten, nachdem sie einmal benutzt wurden, effektiv Teil der Trainingsdaten werden. Andernfalls würden wir agieren wie ein Dozent, der jedes Jahr die gleichen Prüfungs-

fragen verwendet – ein cleverer Student wird leicht herausfinden, dass er sich mit dem Vorlesungsstoff nicht abmühen muss, sondern dass es genügt, sich die Antworten auf genau diese Fragen zu merken.

Wir sollten immer im Blick behalten, dass die verwendete Trainingsmenge eine zufällige Stichprobe ist. Wenn wir also für die gleiche Anwendung noch einmal eine Stichprobe nehmen, wird unsere Datenmenge ein klein wenig anders aussehen, das angepasste h wird etwas anders sein und wir werden einen etwas anderen Validierungsfehler bekommen. Oder wir bekommen für eine feste Datenbasis, die wir in Trainingsdaten, Validierungsdaten und Testdaten unterteilen, unterschiedliche Fehler, je nachdem, wie wir die Unterteilung wählen. Diese kleinen Unterschiede im Fehler erlauben uns eine Abschätzung, wie groß Abweichungen sein sollten, um als signifikant betrachtet zu werden und nicht als Zufall. Das heißt, bei der Wahl zwischen zwei Hypotheseklassen $\mathcal{H}_i$ und $\mathcal{H}_j$ wenden wir beide mehrere Male auf einer Reihe von Trainings- und Validierungsmengen an und prüfen, ob der Unterschied zwischen den mittleren Fehlern von h_i und h_j größer als die mittlere Differenz zwischen verschiedenen h_i ist. In Kapitel 19 diskutieren wir, wie Experimente auf der Basis maschinellen Lernens konstruiert werden können, die mithilfe begrenzter Daten unsere Fragen beantworten – zum Beispiel: Welche ist die beste Hypotheseklasse? – und wie die Ergebnisse dieser Experimente analysiert werden können, so dass wir statistisch signifikante Schlussfolgerungen erhalten, die so wenig wie möglich vom Zufall beeinflusst werden.

2.8 Dimensionen eines Algorithmus für überwachtes Lernen

Fassen wir nun noch einmal zusammen und treffen wir allgemeingültige Aussagen. Wir haben eine Stichprobe

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N . \tag{2.19}$$

UNABHÄNGIG IDENTISCH VERTEILT

Die Stichprobe ist *unabhängig und identisch verteilt* (Abk. i.i.d. für engl. *independent and identically distributed*); die Reihenfolge ist nicht wichtig und alle Instanzen sind aus derselben gemeinsamen Verteilung $p(x, r)$ gezogen worden. t indiziert eine von N Instanzen, x^t ist dann die belieibigdimensionale Eingabe und r^t ist die zugehörige gewünschte Ausgabe. r^t ist entweder 0/1 beim Zweiklassenlernen oder ein K-dimensionaler Binärvektor (bei dem genau eine der Dimensionen 1 ist und alle anderen 0 sind) bei Klassifikationen mit ($K > 2$) Klassen oder ein reeller Wert bei der Regression.

Das Ziel besteht darin, eine gute und nützliche Approximation für r^t unter Verwendung des Modells $g(x^t|\theta)$ zu finden. Dabei gilt es, drei Entscheidungen zu treffen:

1. Hinsichtlich des *Modells*, dass wir beim Lernen nutzen, geschrieben in der Form

   $$g(x|\theta)\,,$$

   wobei $g(\cdot)$ das Modell ist, x für die Eingabe steht und θ die Parameter bezeichnet.

   $g(\cdot)$ definiert die Hypothesenklasse $\mathcal{H}$ und ein bestimmter Wert für θ instanziert eine Hypothese $h \in \mathcal{H}$. Beim Klassenlernen beispielsweise haben wir ein Rechteck als Modell genommen, dessen vier Koordinaten θ bilden. Bei der linearen Regression ist das Modell eine lineare Funktion der Eingabe, deren Anstieg und Achsenabschnitt die von den Daten erlernten Parameter sind. Das Modell (die induktive Verzerrung) oder $\mathcal{H}$ wird durch den Entwickler des maschinellen Lernsystems basierend auf dessen Kenntnis der Anwendung fixiert. Die Parameter werden durch einen Lernalgorithmus abgestimmt, indem ein Trainingsmenge genutzt wird, welche aus dieser Anwendung erhoben wurde. Die Hypothese h wird durch einen Lernalgorithmus gewählt (die Parameter werden eingestellt), wobei die von $p(x, r)$ erhobene Trainingsmenge verwendet wird.

2. Hinsichtlich der *Verlustfunktion* $L(\cdot)$, um die Differenz zwischen gewünschter Ausgabe r^t und unserer Approximation, $g(x^t|\theta)$, für die gegenwärtigen Werte der Parameter θ zu berechnen. Der *Näherungsfehler* oder *Verlust* ist die Summe der Verluste der einzelnen Instanzen.

   $$E(\theta|\mathcal{X}) = \sum_t L(r^t, g(x^t|\theta))\,. \tag{2.20}$$

   Beim Klassenlernen, wenn die Ausgabewerte 0/1 sind, überprüft $L(\cdot)$ auf Gleichheit. Bei der Regression sind die Ausgabewerte numerisch. Das erlaubt uns, Abstände zwischen ihnen zu definieren. Eine Möglichkeit besteht hier darin, das Quadrat der Differenz zu nutzen.

3. Hinsichtlich der *Optimierungsprozedur*, um den Wert θ^* zu finden, der den Approximationsfehler minimiert:

   $$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E(\theta|\mathcal{X})\,, \tag{2.21}$$

   wobei argmin das Argument θ zurückliefert, das $E(\theta|X)$ minimiert. Bei der polynomialen Regression können wir dieses Optimum analytisch bestimmen, doch das ist nicht immer der Fall. Bei anderen Modellen und Fehlerfunktionen wird die Komplexität des Optimierungsproblems wichtig. Insbesondere interessiert uns, ob es ein eindeutiges Minimum gibt, was einer global optimalen Lösung entspricht, oder ob es mehrere Minima und somit lokal optimale Lösungen gibt.

Damit dieser Ansatz gut funktioniert, sollten die nachfolgenden Bedingungen erfüllt sein. Erstens sollte die Klasse der Hypothesen von $g(\cdot)$ groß genug oder von ausreichender Mächtigkeit sein, um die unbekannte Funktion zu beinhalten, welche die verrauschten Daten, repräsentiert durch r^t, erzeugt hat. Zweitens sollte es genügend Trainingsdaten geben, um es uns zu ermöglichen, die korrekte Hypothese (oder eine ausreichend gute) in der Hypothesenklasse ausfindig zu machen. Drittens benötigen wir eine gute Optimierungsmethode, welche die korrekte Hypothese bei gegebenen Daten findet.

Verschiedene maschinelle Lernalgorithmen unterscheiden sich entweder in den Modellen, die sie voraussetzen (ihre Hypothesenklasse/induktive Verzerrung), in den Verlustmaßen, die sie verwenden oder in der Optimierungsprozedur, die bei ihnen zum Einsatz kommt. In den kommenden Kapiteln werden uns diverse Beispiele begegnen.

2.9 Anmerkungen

Mitchell entwickelte Versionsräume und den Algorithmus „Candidate Elimination“, um S und G inkrementell aufzubauen, wenn Instanzen nach und nach bereitgestellt werden. Für eine aktuelle Betrachtung sei auf Mitchell 1997 verwiesen. Das Rechteck-Lernen wurde aus Übung 2.4 in Mitchell (1997) übernommen. Hirsh (1990) diskutiert die Frage, wie verschiedene Versionsräume mit dem Fall umgehen, wenn Instanzen durch geringes Rauschen gestört werden.

In einem der frühesten Werke zum maschinellen Lernen schlug Winston (1975) das Konzept der „Beinaheverfehlung“ („near miss“) vor. Eine Beinaheverfehlung ist ein negatives Beispiel, welches einem positiven Beispiel stark ähnelt. In unserer Terminologie erkennen wir, dass eine Beinaheverfehlung einer Instanz gleichkommt, welche in den Graubereich zwischen S und G fällt und daher den Margin beeinflusst, nützlicher für das Lernen sein würde, als ein herkömmliches positives oder negatives Beispiel. Die Instanzen, die sich nahe an der Grenze befinden, sind es, die die Grenze definieren (oder sie stützen); Instanzen, die im Inneren liegen und von vielen Instanzen mit dem gleichen Label umgeben sind, können entfernt werden, ohne dass dies die Grenze beeinflusst.

Verwandt mit dieser Idee ist auch das Gebiet des *aktiven Lernens*, bei dem der Lernalgorithmus selbst Instanzen generieren und deren Einteilung anfordern kann, statt sie nur passiv zugeführt zu bekommen (Angluin 1988; siehe Übung 5).

Die VC-Dimension wurde von Vapnik und Chervonenkis in den frühen 1970er Jahren entworfen. Eine neuere Quelle ist Vapnik 1995, in der er schreibt: „Nothing is more practical than a good theory“ (S. X) – was auf das Feld des maschinellen Lernens genauso zutrifft wie auf jeden anderen Wissenschaftszweig. Man sollte sich nicht gleich auf den Rechner

stürzen; viele Stunden von nutzloser Programmierung kann man sich durch ein wenig Denkarbeit, einen Schreibblock und einen Stift ersparen – ein Radiergummi ist jedoch unter Umständen von Nöten.

Das PAC-Modell wurde von Valiant (1984) vorgeschlagen. Die PAC-Analyse zum Lernen eines Rechtecks ist Blumer et al. 1989 entnommen. Ein gutes Lehrbuch zur rechnergestützten Lerntheorie, welches PAC-Lernen und die VC-Dimension behandelt, ist Kearns und Vazirani 1994.

Die Definition des Optimierungsproblems für die Modellanpassung hat in den letzten Jahren große Bedeutung erlangt. Während man sich früher mit lokalen Gradientenabstiegsverfahren zufrieden gegeben hat, die ausgehend von einem zufällig gewählten Anfangszustand zur nächstgelegenen guten Lösung konvergieren, richtet sich das Interesse heute zum Beispiel darauf, die Konvexität des Problems zu beweisen, d. h. die Existenz einer eindeutigen, globalen Lösung (Boyd und Vandenberghe, 2004). Wenn der Umfang der Datenmenge wächst und die Modelle komplexer werden, sind wir außerdem zum Beispiel daran interessiert, wie schnell das Optimierungsverfahren gegen eine Lösung konvergiert.

2.10 Übungen

1. Nehmen wir an, unsere Hypothesenklasse ist die Menge aller Kreise statt aller Rechtecke. Welche Parameter liegen vor? Wie können die Parameter einer Kreishypothese in so einem Fall berechnet werden? Was passiert, wenn es sich um eine Ellipse handelt? Warum ist es sinnvoller, eine Ellipse statt eines Kreises zu nutzen?

   Lösung: Im Falle eines Kreises sind die Parameter der Mittelpunkt und der Radius (siehe Abbildung 2.11). Wir müssen dann S und G so bestimmen, dass S den kleinsten Kreis bildet, der alle positiven Beispiele enthält, und G muss der größte Kreis sein, der alle positiven, jedoch kein negatives Beispiel enthält. Jeder zwischen beiden liegende Kreis ist eine konsistente Hypothese.

   Es ist sinnvoller, eine Ellipse zu verwenden, da die beiden Achsen nicht die gleiche Skala verwenden müssen. Eine Ellipse hat zwei separate Parameter für die Längen der beiden Achsen anstatt einen einzigen wie im Falle des Kreises (den Radius). In der Tat sind Preis und Motorleistung positiv korreliert; der Preis eines Autos steigt tendenziell, wenn die Motorleistung zunimmt, und daher ist es zweckmäßig, eine geneigte Ellipse zu verwenden. Solche Modelle werden wir in Kapitel 5 kennenlernen.

2. Stellen Sie sich vor, Ihre Hypothese ist kein Rechteck, sondern die Vereinigung von zwei (oder $m > 1$) Rechtecken. Worin liegt der Vorteil solch einer Hypothesenklasse? Zeigen Sie, dass jede beliebige Klasse durch solch eine Hypothesenklasse mit ausreichend großem m dargestellt werden kann.

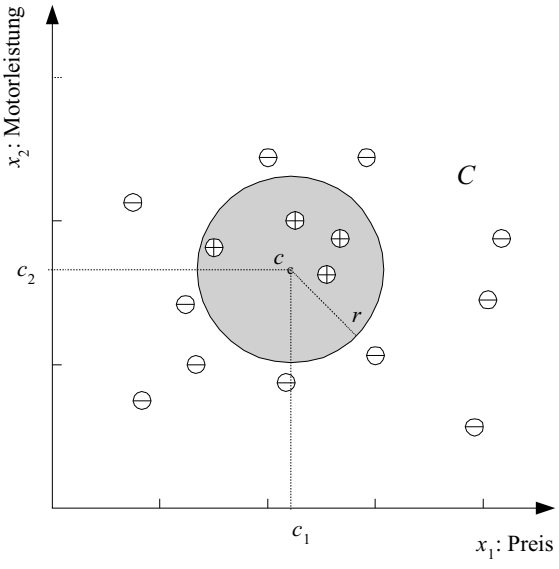


***Abb. 2.11*:** Die Hypothesenklasse ist ein Kreis, der durch zwei Parameter – die Mittelpunktskoordinaten und den Radius – charakterisiert ist.

LÖSUNG: Im Falle eines einzelnen Rechtecks sollten alle Instanzen eine einzige Gruppe bilden. Bei zwei Rechtecken (siehe Abbildung 2.12) können die positiven Instanzen zwei, möglicherweise disjunkte, Cluster im Eingaberaum bilden. Beachten Sie, dass jedes Rechteck einer Konjunktion der beiden Eingabeattribute entspricht; mehrere Rechtecke zu haben, entspricht einer Disjunktion. Jede logische Aussage kann als eine Disjunktion von Konjunktionen geschrieben werden. Im ungünstigsten Fall ($m = N$) haben wir für jede Instanz ein separates Rechteck.

3. Bei vielen Anwendungen führen falsche Entscheidungen – falsch positive wie auch falsch negative – zu monetären Kosten, wobei die Kosten für falsch positive und falsch negative Entscheidungen unterschiedlich sein können. Wo sollte h unter Berücksichtigung der relativen Kosten von S und G platziert werden?

   LÖSUNG: Wir können sehen, dass es in S keine falsch positiven Entscheidungen gibt, sondern nur falsch negative. Umgekehrt gibt es in G keine falsch negativen, sondern nur falsch positive Entscheidungen. Wenn falsch positive und falsch negative Entscheidungen gleich schlecht sind, sollten wir daher h in der Mitte platzieren; wenn falsch positive Entscheidungen höhere Kosten haben, sollte h näher bei S liegen; wenn falsch negative Entscheidungen teurer sind, sollte h näher bei G liegen.

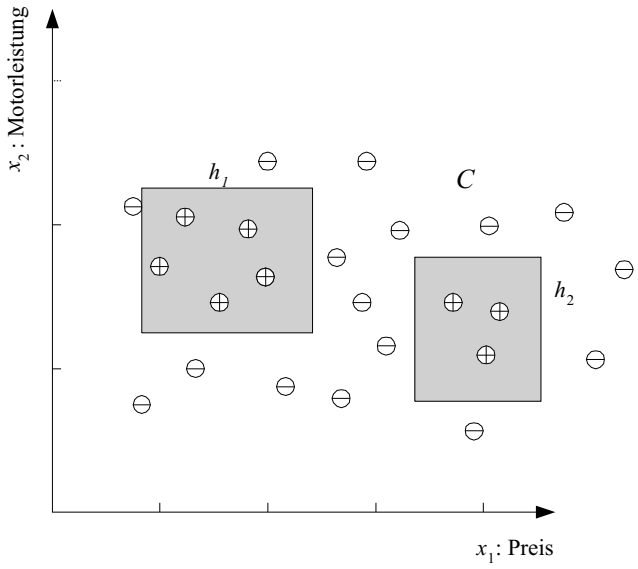


***Abb. 2.12*:** Die Hypothesenklasse entspricht der Vereinigungsmenge der beiden Rechtecke.

4. Die Komplexität der meisten Lernalgorithmen ist eine Funktion der Trainingsmenge. Schlagen Sie einen Filteralgorithmus vor, der redundante Instanzen findet.

   LÖSUNG: Die Instanzen, die die Hypothese beeinflussen, sind jene, welche in der Umgebung von Instanzen mit einem anderen Label liegen. Eine positive Instanz, die auf allen Seiten von weiteren positiven Instanzen umgeben ist, wird nicht benötigt; ebenso wenig eine negative Instanz, die von vielen negativen Instanzen umgeben ist. Wir werden solche nachbarbasierten Methoden in Kapitel 8 diskutieren.

5. Wenn wir einen Supervisor haben, der uns die Einordnung für jedes beliebige $\boldsymbol{x}$ mitteilen kann, wo sollte dann $\boldsymbol{x}$ unserer Wahl nach liegen, damit wir mit weniger Anfragen lernen können?

   LÖSUNG: Das Gebiet der Nichteindeutigkeit liegt zwischen S und G. Am besten wäre es daher, Anfragen für dieses Gebiet zu haben, so dass wir es kleiner machen können. Wenn sich eine gegebene Instanz dort als positiv erweist, können wir S um diese Instanz größer machen; wenn sie negativ ist, können wir G bis zu ihr verkleinern.

6. In Gleichung 2.13 haben wir die Quadrate der Differenzen zwischen tatsächlichem und geschätzten Wert summiert. Diese Fehlerfunktion ist eine der am häufigsten genutzten aber nur eine von mehreren

möglichen Fehlerfunktionen. Da sie die Quadrate der Differenzen summiert, ist sie nicht immun gegenüber Ausreißern. Wie sähe eine bessere Fehlerfunktion aus, mit der eine *robuste Regression* implementiert werden kann?

7. Leiten Sie Gleichung 2.17 ab.

8. Nehmen Sie an, unsere Hypothesenklasse ist eine Menge an Geraden. Nehmen Sie weiterhin an, wir nutzen eine Gerade, um die positiven und negativen Beispiele zu trennen statt die positiven Beispiele mit einem Rechteck abzugrenzen, so dass die negativen außerhalb liegen (siehe Abbildung 2.13). Zeigen Sie, dass die VC-Dimension einer Geraden gleich 3 ist.

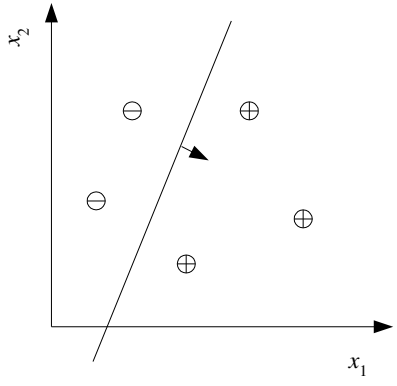


***Abb. 2.13*:** Eine Gerade, die positive und negative Instanzen trennt.

9. Zeigen Sie, dass die VC-Dimension der Hypothesenklasse aller Dreiecke gleich 7 in zwei Dimensionen ist. (Hinweis: Um die bestmögliche Trennung zu erreichen, ist es ratsam, die sieben Punkte äquidistant auf einem Kreis zu platzieren.)

10. Nehmen Sie wie in Übung 8 an, dass unsere Hypothesenklasse eine Menge von Geraden ist. Schreiben Sie eine Fehlerfunktion auf, die nicht nur die Anzahl der falschen Zuordnungen minimiert, sondern gleichzeitig den Margin maximiert.

11. Eine Quelle des Rauschens sind Fehler bei den Labels. Schlagen Sie ein Verfahren vor, mit dem sich Datenpunkte finden lassen, die mit großer Wahrscheinlichkeit falsche Labels haben.

2.11 Literaturangaben

Angluin, D. 1988. „Queries and Concept Learning.“ *Machine Learning* 2: 319–342.

Blumer, A., A. Ehrenfeucht, D. Haussler and M. K. Warmuth. 1989. „Learnability and the Vapnik-Chervonenkis Dimension.“ *Journal of the ACM* 36: 929–965.

Boyd, S., and L. Vandenberge. 2004. *Convex Optimization.* Cambridge. UK: Cambridge University Press.

Dietterich, T. G. 2003. „Machine Learning.“ In *Nature Encyclopedia of Cognitive Science.* London: Macmillan.

Hirsh, H. 1990. *Incremental Version Space Merging: A General Framework for Concept Learning.* Boston: Kluwer.

Kearns, M. J. and U. V. Vazirani. 1994. *An Introduction to Computational Learning Theory.* Cambridge, MA: The MIT Press.

Mitchell, T. 1997. *Machine Learning.* New York: McGraw-Hill.

Valiant, L. 1984. „A Theory of the Learnable.“ *Communications of the ACM* 27: 1134–1142.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Winston, P. H. 1975. „Learning Structural Descriptions from Examples.“ In *The Psychology of Computer Vision,* ed. P. H. Winston, 157–209. New York: McGraw-Hill.

3 Bayessche Entscheidungstheorie

Wir besprechen die Wahrscheinlichkeitstheorie als Gesamtkonzept innerhalb dessen Entscheidungen unter Ungewissheit getroffen werden. Bei der Klassifikation wird der Satz von Bayes genutzt, um die Wahrscheinlichkeiten der Klassen zu berechnen. Wir verallgemeinern dies, um zu diskutieren, wie wir rationale Entscheidungen treffen können, um das erwartete Risiko zu minimieren. Wir stellen ebenfalls Bayessche Netze vor, um visuell und effizient Abhängigkeiten zwischen Zufallsvariablen darzustellen. Außerdem behandeln wir das Lernen von Assoziationsregeln anhand von Daten.

3.1 Einführung

Die Programmierung von Rechnern, um aus Daten Rückschlüsse zu ziehen, ist eine Mischung aus Methoden der Statistik und der Informatik, bei der die Statistiker den mathematischen Rahmen für die Dateninferenz liefern und Informatiker an einer effizienten Implementierung für die Inferenzmethoden arbeiten.

Die Daten stammen aus einem Prozess, der nicht vollständig bekannt ist. Für diesen Mangel an Kenntnis wird durch die Modellierung des Prozesses als Zufallsprozess Rechnung getragen. Der Prozess mag vielleicht deterministisch sein, aber da wir keine komplette Kenntnis über ihn besitzen, modellieren wir ihn als zufällig und nutzen die Wahrscheinlichkeitstheorie, um ihn zu analysieren. An dieser Stelle sei ein kurzer Blick in den Anhang nahegelegt, um sich grundlegende Wahrscheinlichkeitstheorien noch einmal ins Gedächtnis zu rufen, bevor mit diesem Kapitel fortgefahren wird.

Das Werfen einer Münze ist ein Zufallsprozess, weil wir für keinen der Münzwürfe vorhersagen können, ob das Ergebnis Kopf oder Zahl sein wird – deswegen werfen wir Münzen oder kaufen Lotterielose oder schließen Versicherungen ab. Wir können nur von der Wahrscheinlichkeit sprechen, mit der beim nächsten Münzwurf entweder Kopf oder Zahl fallen wird. Man könnte nun argumentieren, dass wir bei Verfügbarkeit zusätzlichen Wissens – beispielsweise hinsichtlich der genauen Zusammensetzung der Münze, ihrer Ausgangsposition, der Kraft und ihrer Wirkung, die beim

Werfen auf die Münze ausgeübt wird, wo und wie sie aufgefangen wird und so weiter – das exakte Ergebnis des Münzwurfes vorhersagen könnten.

UNBEOBACHTBARE VARIABLE

BEOBACHTBARE VARIABLE

Diese zusätzlichen Informationen, auf die wir keinen Zugriff haben, nennt man die *unbeobachtbaren Variablen*. Für das Beispiel des Münzwurfs ist die einzige *beobachtbare Variable* das Ergebnis des Wurfs. Wenn wir die Werte der unbeobachtbaren Variablen mit $\boldsymbol{z}$ und die Werte der beobachtbaren mit x bezeichnen, dann haben wir in der Realität die Formel

$$x = f(\boldsymbol{z}) ,$$

wobei $f(\cdot)$ die deterministische Funktion ist, welche das Ergebnis aus den unbeobachtbaren Informationen definiert. Weil wir den Prozess auf diese Weise nicht modellieren können, definieren wir das Ergebnis X als Zufallsvariable, deren Realisierungen nach einer Wahrscheinlichkeitsverteilung $P(X = x)$, die den Prozess spezifiziert, gezogen wurden.

Das Ergebnis des Münzwurfs ist Kopf oder Zahl und wir definieren eine Zufallsvariable, die einen von zwei Werten annehmen kann. Gehen wir davon aus, dass $X = 1$ für das Ergebnis Kopf und $X = 0$ für das Ergebnis Zahl steht. Derartige X unterliegen einer Bernoulli-Verteilung, wobei der Parameter der Verteilung p_o die Wahrscheinlichkeit dafür ist, dass das Ergebnis Kopf ist.

$$P(X = 1) = p_o \text{ und } P(X = 0) = 1 - P(X = 1) = 1 - p_o .$$

Angenommen, wir sollten das Ergebnis des nächsten Münzwurfs vorhersagen. Wenn wir p_o kennen, wird unsere Vorhersage Kopf sein, wenn $p_o > 0{,}5$ und ansonsten Zahl. Der Grund ist der, dass wir, wenn wir den wahrscheinlicheren Fall wählen, die Fehlerwahrscheinlichkeit, die sich aus 1 minus die Wahrscheinlichkeit unserer Wahl ergibt, minimieren. Wenn es sich um eine ideale Münze mit $p_o = 0{,}5$ handelt, haben wir keine andere Wahl bei der Vorhersage als entweder fortwährend Kopf zu wählen oder selbst eine ideale Münze zu werfen!

STICHPROBE

Wenn wir $P(X)$ nicht kennen und anhand einer gegebenen Stichprobe berechnen wollen, betreten wir die Sphären der Statistik. Wir haben eine *Stichprobe* $\mathcal{X}$ mit Beispielen, die nach der Wahrscheinlichkeitsverteilung der beobachtbaren x^t gezogen wurden; man schreibt dies als $p(x)$. Das Ziel besteht darin, anhand der Stichprobe $\mathcal{X}$ eine Näherungsfunktion $\hat{p}(x)$ für die Verteilung zu definieren.

Für das Beispiel Münzwurf enthält die Stichprobe die Ergebnisse der vergangenen N Würfe. Wenn wir dann $\mathcal{X}$ nutzen, können wir p_o schätzen, was dem Parameter entspricht, der eindeutig die Verteilung beschreibt. Unsere Schätzung von p_o lautet

$$\hat{p}_o = \frac{\#\{\text{Würfe mit Ergebnis Kopf}\}}{\#\{\text{Würfe}\}} .$$

Mit numerischen Zufallsvariablen ist x^t gleich 1, wenn das Ergebnis des Münzwurfs t Kopf ist; andernfalls ist es gleich 0. Für eine Stichprobe der Art {Kopf, Kopf, Kopf, Zahl, Kopf, Zahl, Zahl, Kopf, Kopf} erhalten wir $\mathcal{X} = \{1, 1, 1, 0, 1, 0, 0, 1, 1\}$ und die Schätzung ist

$$\hat{p}_o = \frac{\sum_{t=1}^{N} x^t}{N} = \frac{6}{9} .$$

3.2 Klassifikation

In Abschnitt 1.2.2 haben wir uns mit der Überprüfung von Kunden auf ihre Kreditwürdigkeit befasst und konnten beobachten, dass für eine Bank einige Kunden aufgrund ihrer vergangenen Transaktionen ein niedriges Risiko hinsichtlich ihrer Darlehensrückzahlungen darstellen und die Bank von ihnen profitieren kann, wohingegen andere Kunden ein höheres Risiko darstellen, da sie bereits mit Zahlungen in Verzug gerieten. Bei der Analyse der Daten möchten wir gerne die Klasse der „Kunden mit hohem Risiko“ erlernen, so dass wir in Zukunft bei einem neuen Darlehensantrag prüfen können, ob die jeweilige Person in diese Klasse fällt oder nicht und dementsprechend den Antrag ablehnen oder annehmen. Unter Zuhilfenahme unserer Kenntnis des Antrags gehen wir nun einmal davon aus, dass es zwei beobachtbare Informationen gibt. Wir beobachten diese, weil wir Grund zur Annahme haben, dass sie uns eine Vorstellung von der Kreditwürdigkeit eines Kunden geben. Nehmen wir beispielsweise an, wir beobachten das jährliche Einkommen und die Ersparnisse eines Kunden und stellen dies mit Hilfe zweier Zufallsvariablen X_1 und X_2 dar.

Man könnte wiederum behaupten, dass wir mit Zugang zu weiteren Informationen wie detaillierten Kenntnissen der Wirtschaftslage und umfassendem Wissen über den Kunden, also dessen Absichten, Moralvorstellungen und so weiter, deterministisch berechnen könnten, ob jemand ein Kunde mit hohem oder niedrigem Risiko ist. Dies sind jedoch keine beobachtbaren Variablen. Mit dem, was wir tatsächlich beobachten können, kann die Kreditwürdigkeit eines Kunden durch eine zufällige Bernoulli-Variable $\mathcal{C}$ dargestellt werden, bedingt durch die beobachtbaren Variablen $\boldsymbol{X} = [X_1, X_2]^T$, wobei $\mathcal{C} = 1$ einen Kunden mit hohem Risiko und $\mathcal{C} = 0$ einen mit niedrigem Risiko repräsentiert. Wenn wir also $P(\mathcal{C}|X_1, X_2)$ kennen, können wir bei Erhalt eines neuen Antrags mit $X_1 = x_1$ und $X_2 = x_2$ wie folgt vorgehen:

$$\text{wähle} \begin{cases} \mathcal{C} = 1 \text{ falls } P(\mathcal{C} = 1|x_1, x_2) > 0{,}5 , \\ \mathcal{C} = 0 \text{ andernfalls} \end{cases}$$

oder äquivalent dazu

$$\text{wähle} \begin{cases} \mathcal{C} = 1 \text{ falls } P(\mathcal{C} = 1|x_1, x_2) > P(\mathcal{C} = 0|x_1, x_2) , \\ \mathcal{C} = 0 \text{ andernfalls} . \end{cases} \tag{3.1}$$

$1 - \max(P(\mathcal{C} = 1|x_1, x_2), P(\mathcal{C} = 0|x_1, x_2))$ stellt die Fehlerwahrscheinlichkeit dar. Das Beispiel ähnelt dem des Münzwurfs mit der Ausnahme, dass hier die zufällige Bernoulli-Variable $\mathcal{C}$ von zwei anderen beobachtbaren Variablen abhängt. Bezeichnen wir mit $\boldsymbol{x}$ den Vektor der Werte der beobachteten Variablen $\boldsymbol{x} = [x_1, x_2]^T$. Das Problem besteht dann darin, $P(\mathcal{C}|\boldsymbol{x})$ berechnen zu können. Wenn wir den *Satz von Bayes* zu Hilfe nehmen, ergibt sich:

Satz von Bayes

$$P(\mathcal{C}|\boldsymbol{x}) = \frac{P(\mathcal{C})p(\boldsymbol{x}|\mathcal{C})}{p(\boldsymbol{x})} . \tag{3.2}$$

a-priori-Wahrscheinlichkeit

$P(\mathcal{C} = 1)$ nennt man dabei die *a-priori-Wahrscheinlichkeit* dafür, dass $\mathcal{C}$ den Wert 1 annimmt, was in unserem Beispiel der Wahrscheinlichkeit dafür entspricht, dass ein Kunde ein hohes Risiko darstellt, unabhängig vom Wert $\boldsymbol{x}$ – sie beschreibt den Anteil der Hochrisikokunden in unserem Kundenstamm.. Die Bezeichnung a-priori-Wahrscheinlichkeit trägt sie deshalb, weil sie das Wissen darstellt, welches wir über den Wert $\mathcal{C}$ haben, *bevor* wir die beobachtbaren Variablen $\boldsymbol{x}$ betrachten, und es gilt:

$$P(\mathcal{C} = 0) + P(\mathcal{C} = 1) = 1 .$$

Klassen-Likelihood

$p(\boldsymbol{x}|\mathcal{C})$ wird als *Klassen-Likelihood* bezeichnet und ist die bedingte Wahrscheinlichkeit dafür, dass ein zu $\mathcal{C}$ zugehöriges Ereignis den zugehörigen Beobachtungswert $\boldsymbol{x}$ hat. In unserem Fall ist $p(x_1, x_2|\mathcal{C} = 1)$ die Wahrscheinlichkeit dafür, dass für einen Kunden mit hohem Risiko $X_1 = x_1$ und $X_2 = x_2$ zutrifft. Dies sind die Informationen, die wir von den Daten hinsichtlich der Klasse erhalten.

Evidenz

$p(\boldsymbol{x})$, die *Evidenz*, ist die Randwahrscheinlichkeit dafür, dass eine Beobachtung $\boldsymbol{x}$ gemacht wird, unabhängig davon, ob es sich um ein positives oder negatives Beispiel handelt.

$$p(\boldsymbol{x}) = p(\boldsymbol{x}|\mathcal{C} = 1)P(\mathcal{C} = 1) + p(\boldsymbol{x}|\mathcal{C} = 0)P(\mathcal{C} = 0) . \tag{3.3}$$

a-posteriori-Wahrscheinlichkeit

Indem wir den Satz von Bayes nutzen und die a-priori-Wahrscheinlichkeit mit dem, was wir anhand der Daten wissen, kombinieren, berechnen wir die *a-posteriori-Wahrscheinlichkeit* der Klasse, $P(\mathcal{C}|\boldsymbol{x})$, *nachdem* wir die Beobachtung $\boldsymbol{x}$ gemacht haben.

$$\text{a-posteriori-Wahrscheinlichkeit} = \frac{\text{a-priori-Wahrscheinlichkeit} \times \text{Wahrscheinlichkeit}}{\text{Evidenz}}$$

Durch die Normalisierung durch die Evidenz summieren sich die a-Posteriori-Werte zu 1:

$$P(\mathcal{C} = 0|\boldsymbol{x}) + P(\mathcal{C} = 1|\boldsymbol{x}) = 1 .$$

Sobald wir die a-posteriori-Wahrscheinlichkeiten berechnet haben, treffen wir mittels Gleichung 3.1 eine Entscheidung. Für den Augenblick nehmen wir an, dass wir die a-priori-Wahrscheinlichkeiten und die Likelihoods kennen; in späteren Kapiteln werden wir noch diskutieren, wie man $P(\mathcal{C})$ und $p(\boldsymbol{x}|\mathcal{C})$ aus einer gegebenen Stichprobe berechnen kann.

Im allgemeinen Fall haben wir K sich gegenseitig ausschließende Klassen $\mathcal{C}_i, i = 1, \ldots, K$. Bei der optischen Ziffernerkennung beispielsweise besteht die Eingabe aus einem Bitmap-Bild und es gibt zehn Klassen. Die vorliegenden a-priori-Wahrscheinlichkeiten erfüllen

$$P(\mathcal{C}_i) \geq 0 \text{ und } \sum_{i=1}^{K} P(\mathcal{C}_i) = 1 \,. \tag{3.4}$$

$p(\boldsymbol{x}|\mathcal{C}_i)$ ist die Wahrscheinlichkeit dafür, $\boldsymbol{x}$ als Eingabewert zu beobachten, wenn bekannt ist, dass es zur Klasse $\mathcal{C}_i$ gehört. Die a-posteriori-Wahrscheinlichkeit der Klasse $\mathcal{C}_i$ kann berechnet werden als:

$$P(\mathcal{C}_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i)}{\sum_{k=1}^{K} p(\boldsymbol{x}|\mathcal{C}_k)P(\mathcal{C}_k)} \,. \tag{3.5}$$

Für den Minimalfehler selektiert der *Bayessche Klassifikator* die Klasse mit der höchsten a-posteriori-Wahrscheinlichkeit, das heißt, BAYESSCHER KLASSIFIKATOR

$$\text{wähle } \mathcal{C}_i \text{ falls } P(\mathcal{C}_i|\boldsymbol{x}) = \max_k P(\mathcal{C}_k|\boldsymbol{x}) \,. \tag{3.6}$$

3.3 Verluste und Risiken

Es mag der Fall auftreten, dass Entscheidungen nicht gleich gut oder kostspielig sind. Ein Finanzinstitut sollte beim Entscheidungsprozess bezüglich eines eventuellen Darlehens sowohl den potenziellen Gewinn als auch den Verlust in Betracht ziehen. Ein positiv eingeschätzter Kunde mit niedrigem Risiko erhöht den Profit, wohingegen ein negativ bewerteter Kunde mit hohem Risiko den Verlust verringert. Der Verlust für einen fälschlicherweise positiv eingestuften Kunden mit hohem Risiko kann sich vom potenziellen Gewinn für einen fälschlicherweise abgewiesenen Kunden mit niedrigem Risiko unterscheiden. Die Situation ist in anderen Anwendungsgebieten noch kritischer und alles andere als symmetrisch, beispielsweise in der medizinischen Diagnostik oder bei der Erdbebenvorhersage.

Definieren wir die Aktion α_i als die Entscheidung, die Eingabe der Klasse $\mathcal{C}_i$ zuzuweisen, und λ_{ik} sei der durch die Ausführung von Aktion α_i entstandene *Verlust*, falls die Eingabe eigentlich zu $\mathcal{C}_k$ gehört. In diesem VERLUSTFUNKTION

ERWARTETES RISIKO

Fall ergibt sich das *erwartete Risiko* für die Ausführung von Aktion α_i aus

$$R(\alpha_i|\boldsymbol{x}) = \sum_{k=1}^{K} \lambda_{ik} P(\mathcal{C}_k|\boldsymbol{x}) \tag{3.7}$$

und wir wählen die Aktion mit minimalem Risiko:

$$\text{wähle } \alpha_i, \text{ falls } R(\alpha_i|\boldsymbol{x}) = \min_k R(\alpha_k|\boldsymbol{x}) \,. \tag{3.8}$$

0/1-VERLUST

Definieren wir nun K Aktionen $\alpha_i, i = 1, \ldots, K$, wobei α_i aus der Zuweisung von $\boldsymbol{x}$ zu C_i besteht. Im speziellen Fall des *0/1-Verlusts* mit

$$\lambda_{ik} = \begin{cases} 0 \text{ falls } i = k \,, \\ 1 \text{ falls } i \neq k \end{cases} \tag{3.9}$$

sind alle korrekten Entscheidungen verlustfrei und alle Fehler gleich kostspielig. Das Risiko für die Ausführung von Aktion α_i ergibt sich aus

$$\begin{aligned} R(\alpha_i|\boldsymbol{x}) &= \sum_{k=1}^{K} \lambda_{ik} P(\mathcal{C}_k|\boldsymbol{x}) \\ &= \sum_{k \neq i} P(\mathcal{C}_k|\boldsymbol{x}) \\ &= 1 - P(\mathcal{C}_i|\boldsymbol{x}) \,, \end{aligned}$$

weil $\sum_k P(\mathcal{C}_k|\boldsymbol{x}) = 1$. Um also das Risiko zu minimieren, wählen wir den wahrscheinlichsten Fall. In späteren Kapiteln werden wir aus Gründen der Einfachheit immer von diesem Fall ausgehen und die Klasse mit der höchsten a-posteriori-Wahrscheinlichkeit wählen. Man beachte jedoch, dass es sich hierbei wirklich um einen speziellen Fall handelt und Anwendungsszenarien nur selten einem symmetrischen 0/1-Verlust unterliegen. Im allgemeinen Fall ist es ein einfacher Nachbearbeitungsschritt von den a-posteriori-Werten zu den Risiken, um dann die entsprechende Aktion zur Minimierung des Risikos auszuführen.

Bei einigen Anwendungen können falsche Entscheidungen, sprich Fehlklassifikationen, schwere Folgen haben, und es ist normalerweise erforderlich, dass eine komplexere, beispielsweise eine manuelle Entscheidung, getroffen wird, wenn das automatische System die Entscheidung nur mit niedriger Sicherheit liefert. Wenn wir zum Beispiel ein optisches Zifferner-kennungssystem nutzen, um Postleitzahlen auf Briefumschlägen zu lesen, verursacht ein falsches Einlesen einer Zahl das Verschicken des Briefes an eine falsche Adresse.

ABLEHNUNG

In solch einem Fall definieren wir eine zusätzliche Aktion α_{K+1} für *Ablehnung* oder *Zweifel* mit $\alpha_i, i = 1, \ldots, K$ als übliche Entscheidungsaktionen hinsichtlich der Klassen $\mathcal{C}_i, i = 1, \ldots, K$ (Duda, Hart und Stork, 2001).

Eine mögliche Verlustfunktion ergibt sich aus

$$\lambda_{ik} = \begin{cases} 0 \text{ falls } i = k\,, \\ \lambda \text{ falls } i = K+1\,, \\ 1 \text{ andernfalls}\,, \end{cases} \tag{3.10}$$

wobei $0 < \lambda < 1$ den durch die Wahl der $(K+1)$-ten Ablehnungsaktion ausgelösten Verlust repräsentiert. Das Risiko der Ablehnung ist dann

$$R(\alpha_{K+1}|\boldsymbol{x}) = \sum_{k=1}^{K} \lambda P(\mathcal{C}_k|\boldsymbol{x}) = \lambda \tag{3.11}$$

und das Risiko für die Wahl der Klasse $\mathcal{C}_i$ ist

$$R(\alpha_i|\boldsymbol{x}) = \sum_{k \neq i} P(\mathcal{C}_k|\boldsymbol{x}) = 1 - P(\mathcal{C}_i|\boldsymbol{x})\,. \tag{3.12}$$

Die optimale Entscheidungsregel ergibt sich wie folgt:

$$\begin{array}{ll} \text{wähle } \mathcal{C}_i & \text{falls } R(\alpha_i|\boldsymbol{x}) < R(\alpha_k|\boldsymbol{x}) \text{ für alle } k \neq i \text{ und} \\ & R(\alpha_i|\boldsymbol{x}) < R(\alpha_{K+1}|\boldsymbol{x})\,, \\ \text{lehne ab} & \text{falls } R(\alpha_{K+1}|\boldsymbol{x}) < R(\alpha_i|\boldsymbol{x}), i = 1, \ldots, K\,. \end{array} \tag{3.13}$$

In Anbetracht der Verlustfunktion aus Gleichung 3.10 lässt sich vereinfacht sagen:

$$\begin{array}{ll} \text{wähle } \mathcal{C}_i & \text{falls } P(\mathcal{C}_i|\boldsymbol{x}) > P(\mathcal{C}_k|\boldsymbol{x}) \text{ für alle } k \neq i \text{ und} \\ & P(\mathcal{C}_i|\boldsymbol{x}) > 1 - \lambda\,, \\ \text{lehne ab} & \text{andernfalls}\,. \end{array} \tag{3.14}$$

Dieser gesamte Ansatz ist bedeutungsvoll, wenn $0 < \lambda < 1$. Falls $\lambda = 0$, lehnen wir immer ab; eine Ablehnung ist genauso gut wie eine korrekte Klassifikation. Falls $\lambda \geq 1$, lehnen wir niemals ab; eine Ablehnung ist genauso kostspielig oder gar kostspieliger als ein Fehler.

Im Falle der Ablehnung wählen wir zwischen der automatischen Entscheidung, die der Computer trifft, und der menschlichen Entscheidung, die kostspieliger ist, von der jedoch angenommen wird, dass sie mit höherer Wahrscheinlichkeit korrekt ist. Ähnlich können wir uns eine Kaskade aus mehreren automatischen Entscheidern vorstellen, die zunehmend kostspieliger werden und gleichzeitig mit wachsender Wahrscheinlichkeit korrekte Entscheidungen treffen. Wir diskutieren solche Kaskaden in Kapitel 17 im Zusammenhang mit der Kombination mehrerer Lerner.

3.4 Diskriminanzfunktionen

Diskriminanz-funktion

Die Klassifikation kann auch als die Implementierung einer Menge von *Diskriminanzfunktionen*, $g_i(\boldsymbol{x}), i = 1, \ldots, K$, gesehen werden, so dass gilt:

$$\text{wähle } \mathcal{C}_i, \text{ falls } g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x}) \,. \tag{3.15}$$

Wir können auf diese Art und Weise den Bayesschen Klassifikator darstellen, wenn wir

$$g_i(\boldsymbol{x}) = -R(\alpha_i|\boldsymbol{x})$$

setzen und die maximale Diskriminanzfunktion dem minimalen bedingten Risiko entspricht. Wenn wir die 0/1-Verlustfunktion nutzen, erhalten wir

$$g_i(\boldsymbol{x}) = P(\mathcal{C}_i|\boldsymbol{x}) \,.$$

Ignorieren wir den gemeinsamen normalisierenden Term $p(\boldsymbol{x})$, so können wir auch schreiben:

$$g_i(\boldsymbol{x}) = p(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i) \,.$$

Entscheidungs-regionen

Dadurch wird der Merkmalsraum in K *Entscheidungsregionen* $\mathcal{R}_1, \ldots, \mathcal{R}_K$ unterteilt, wobei $\mathcal{R}_i = \{\boldsymbol{x}|g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x})\}$. Die Regionen sind durch *Entscheidungsgrenzen* voneinander getrennt. Dabei handelt es sich um Teile des Merkmalsraums, in denen Pattsituationen unter den größten Diskriminanzfunktionen auftauchen (siehe Abbildung 3.1).

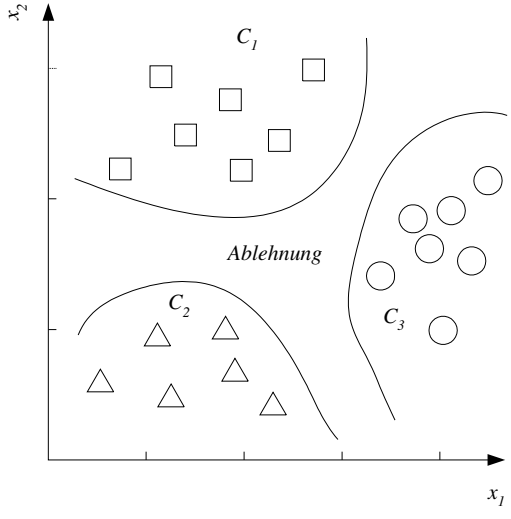


Abb. 3.1: Beispiel für Entscheidungsregionen und Entscheidungsgrenzen.

Wenn es zwei Klassen gibt, können wir eine einzelne Diskriminanzfunktion definieren:

$$g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$$

und es gilt:

$$\text{wähle} \begin{cases} \mathcal{C}_1 \text{ falls } g(\boldsymbol{x}) > 0\,, \\ \mathcal{C}_2 \text{ andernfalls}\,. \end{cases}$$

Ein Beispiel ist ein Zweiklassenlernproblem, bei dem die positiven Beispiele als $\mathcal{C}_1$ und die negativen Beispiele als $\mathcal{C}_2$ gesehen werden können. Wenn $K = 2$, dann handelt es sich bei dem Klassifikationssystem um ein *dichotomes* und ab $K \geq 3$ um ein *polychotomes*.

DICHOTOME UND POLYCHOTOME KLASSIFIKATION

3.5 Assoziationsregeln

Eine *Assoziationsregel* ist eine Beziehung der Form $X \to Y$. Ein Beispiel für solch eine Assoziationsregel bietet die *Warenkorbanalyse*, bei der wir die Abhängigkeit zwischen zwei Artikeln X und Y herausfinden wollen. Eine typische Anwendung findet sich im Warenhandel, wo X und Y verkaufte Artikel darstellen (Abschnitt 1.2.1).

ASSOZIATIONSREGEL

WARENKORB-ANALYSE

Beim Erlernen von Assoziationsregeln werden oft drei Maße berechnet:

- der *Support* der Assoziationsregel $X \to Y$:

SUPPORT

$$\begin{aligned} \text{Support}(X, Y) &\equiv P(X, Y) \\ &= \frac{\#\{\text{Kunden, die } X \text{ und } Y \text{ gekauft haben}\}}{\#\{\text{Kunden}\}} \end{aligned} \tag{3.16}$$

- die *Konfidenz* der Assoziationsregel $X \to Y$:

KONFIDENZ

$$\begin{aligned} \text{Konfidenz}(X \to Y) &\equiv P(Y|X) = \frac{P(X, Y)}{P(X)} \\ &= \frac{\#\{\text{Kunden, die } X \text{ und } Y \text{ gekauft haben}\}}{\#\{\text{Kunden, die } X \text{ gekauft haben}\}} \end{aligned} \tag{3.17}$$

- der *Lift* der Assoziationsregel $X \to Y$

LIFT

$$\text{Lift}(X \to Y) = \frac{P(X, Y)}{P(X)P(Y)} = \frac{P(Y|X)}{P(Y)} \tag{3.18}$$

Es gibt noch weitere Maße (Omiecinski 2003), doch diese drei, vor allem die ersten beiden, sind die bekanntesten und gebräuchlichsten. Die *Konfidenz* ist die bedingte Wahrscheinlichkeit $P(Y|X)$, was dem entspricht, was wir normalerweise berechnen. Um mit genug Sicherheit behaupten zu können, dass eine Regel gilt, sollte der Wert möglichst nahe 1 liegen und deutlich größer als $P(Y)$ sein, also der allgemeinen Wahrscheinlichkeit dafür, dass Kunden Y kaufen. Uns interessiert ebenfalls die Maximierung des *Supports* der Regel, denn selbst bei vorhandener Abhängigkeit mit starker Konfidenz ist die Regel wertlos, wenn die Anzahl an Kunden nur gering ist. Der Support drückt die statistische Signifikanz der Regel aus, wohingegen die Konfidenz die Stärke der Regel repräsentiert. Die Mindestwerte für Support und Konfidenz werden durch die Handelsfirma festgelegt, und in der Datenbasis wird nach allen Regeln mit höheren Werten für Support und Konfidenz gesucht.

Wenn X und Y unabhängig sind, dann erwarten wir, dass der Lift nahe bei eins liegt. Wenn das Verhältnis $P(Y|X)/P(Y)$ von eins abweicht – d. h., wenn die Wahrscheinlichkeiten $P(Y|X)$ und $P(Y)$ verschieden sind, dann muss es eine Abhängigkeit zwischen den Artikeln X und Y geben. Ist der Lift größer als eins, dann können wir sagen, dass Artikel X Artikel Y wahrscheinlicher macht; ist er kleiner als eins, wird Y durch X unwahrscheinlicher.

Diese Formeln können leicht auf den Fall von mehr als zwei Artikeln verallgemeinert werden. Beispielsweise könnten wir für eine Menge $\{X, Y, Z\}$ aus drei Artikeln nach einer Regel wie $X, Z \rightarrow Y$ suchen, also nach $P(Y|X, Z)$. Wir sind daran interessiert, alle solchen Regeln mit ausreichend großen Support- und Konfidenzwerten zu finden, und da eine Verkaufsdatenbank normalerweise sehr groß ist, wollen wir die Abhängigkeiten mit möglichst wenigen Durchläufen der Datenbank finden. Es existiert ein effizienter Algorithmus, der sogenannte *Apriori*-Algorithmus (Agrawal et al. 1996), der genau dies tut. Er besteht aus zwei Schritten: (1) Identifizieren häufig auftretender Artikelmengen, also solchen die ausreichend Support haben, und (2) Umwandeln dieser Artikelmengen in Regeln mit ausreichender Konfidenz. Der zweite Schritt erfolgt dadurch, dass die in einer Artikelmenge enthaltenen Artikel in zwei Gruppen unterteilt werden: vorrangige und nachrangige.

Apriori-Algorithmus

1. Um häufig auftretende Artikelmengen schnell zu finden (ohne Auswertung aller möglichen Kombinationen von Artikeln), nutzt der Apriori-Algorithmus die Tatsache aus, dass es für das häufige Auftreten (ausreichenden Support) der Menge $\{X, Y, Z\}$ nötig ist, dass alle ihre Teilmengen, also $\{X, Y\}$, $\{X, Z\}$ und $\{Y, Z\}$, ebenfalls häufig auftreten – das Hinzufügen eines weiteren Artikels kann den Support niemals vergrößern. Somit müssen wir zum Überprüfen der Mengen aus drei Artikeln lediglich alle ihre häufigen Teilmengen aus zwei Artikeln überprüfen. Mit anderen Worten, wenn wir von einer Menge aus zwei Artikeln bereits wissen, dass sie nicht häufig

ist, dann können wir alle ihre Obermengen verwerfen und müssen sie nicht mehr prüfen.

Wir beginnen, indem wir zunächst alle häufigen Mengen mit nur einem Artikel bestimmen. Anschließend erzeugen wir induktiv aus den häufigen k-Artikelmengen die in Frage kommenden $(k+1)$-Artikelmengen und prüfen die so erhaltenen Daten darauf, ob sie genug Support haben. Der Apriori-Algorithmus speichert die häufigen Artikelmengen in einer Hash-Tabelle, auf die leicht zugegriffen werden kann. Die Anzahl der in Frage kommenden (und somit zu prüfenden) Artikelmengen fällt mit zunehmendem k sehr schnell ab. Wenn die größte Artikelmenge aus n Artikeln besteht, dann brauchen wir insgesamt $n+1$ Durchläufe.

2. Nachdem wir die häufigen k-Artikelmengen gefunden haben, müssen wir aus ihnen Regeln ableiten, indem wir die k Artikel in zwei Gruppen – vorrangige und nachrangige Artikel – unterteilen. Wie beim Identifizieren der Artikelmengen starten wir wieder mit einem einzigen Artikel, den wir als nachrangig betrachten, während alle übrigen Artikel die vorrangige Gruppe bilden. Dann prüfen wir für jeden möglichen nachrangigen Artikel, ob er genug Konfidenz hat. Wenn nicht, wird er entfernt.

   Man beachte, dass es für ein und dieselbe Artikelmenge mehrere Regeln mit unterschiedlichen Gruppen von vorrangigen und nachrangigen Artikeln geben kann. Wir prüfen nun induktiv, ob wir einen weiteren Artikel aus der Gruppe der vorrangigen in die der nachrangigen Artikel verschieben können. Regeln, die mehr Artikel in die nachrangige Gruppe überführen, sind spezifischer und nützlicher. Hierbei nutzen wir – wie schon beim Generieren der Artikelmengen – die Tatsache aus, dass für Regeln, die zwei nachrangige Artikel mit ausreichender Konfidenz liefern, beide Regeln mit nur einem nachrangigen Artikel selbst ausreichende Konfidenz haben müssen. Das heißt, wir gehen von Regeln mit einem nachrangigen Artikel zu Regeln mit zwei nachrangigen Artikeln über und müssen nicht alle möglichen Fälle mit zwei nachrangigen Artikeln prüfen (siehe Übung 9).

   Denken Sie daran, dass man aus einer Regel $X \to Y$ nicht auf Kausalität schließen kann, sondern dass es dabei nur um Assoziation geht. Bei einem Problem können auch *verborgene Variablen* auftreten, deren Werte unbekannt bleiben. Die Verwendung von verborgenen Variablen hat den Vorteil, dass die Abhängigkeitsstruktur leichter definiert werden kann. Betrachten wir zum Beispiel eine Warenkorbanalyse, durch die wir die Abhängigkeiten zwischen den verkauften Artikeln aufdecken wollen. Sagen wir, wir wissen, dass es eine Abhängigkeit zwischen „Babynahrung“, „Windeln“ und „Milch“ gibt – in dem Sinne, dass ein Kunde, der einen dieser Artikel kauft, mit größerer Wahrscheinlichkeit auch die anderen beiden kauft.

VERBORGENE VARIABLEN

Anstatt Abhängigkeiten zwischen den drei Artikeln zu formulieren, können wir eine Verbindung unterstellen, nämlich „Baby zu Hause“, die der verborgene Grund für den Konsum dieser drei Artikel ist. Graphische Modelle, wie wir sie in Kapitel 14 behandeln, erlauben es uns, solche verborgenen Variablen zu repräsentieren. Wenn es verborgene Verbindungen gibt, dann werden ihre Werte aus den gegebenen Werten der bekannten Verbindungen geschätzt und eingefügt.

3.6 Anmerkungen

Der Prozess der Entscheidungsfindung unter Ungewissheit blickt auf eine lange Geschichte zurück, und im Laufe der Zeit hat die Menschheit schon an allen möglichen seltsamen Stellen nach Beweisen gesucht, welche die Ungewissheit eliminieren könnten: in den Sternen, in Kristallkugeln, und sogar im Kaffeesatz. Das Schlussfolgern anhand bedeutungsvoller Beweise mit Hilfe der Wahrscheinlichkeitstheorie ist nur wenige Jahrhunderte alt. Für einen geschichtlichen Abriss der Wahrscheinlichkeitsrechnung und Statistik sei Newman 1988 empfohlen sowie die sehr frühen Berichte von Laplace, Bernoulli und anderen, die diese Theorie mitbegründet haben.

Russell und Norvig (2009) bieten eine exzellente Abhandlung zur Nutzwertanalyse und zum Informationswert, in der sie ebenfalls die Darstellung des Nutzwerts in ökonomischen Dimensionen behandeln. Shafer und Pearl (1990) offerieren eine frühe Sammlung von Artikeln zum Schlussfolgern unter Ungewissheit.

Assoziationsregeln werden in vielen Data-Mining-Anwendungen erfolgreich eingesetzt, und wir begegnen ihnen auf vielen Websites, die Bücher, Filme, Musik und anderes mehr empfehlen. Der Algorithmus ist sehr einfach, und seine effiziente Implementierung auf sehr großen Datenbasen ist kritisch (Zhang und Zhang 2002; Li 2006). In Kapitel 14 werden wir sehen, wie sich Assoziationsregeln zu Konzepten verallgemeinern lassen, die nicht notwendigerweise binär sind und Assoziationen unterschiedlichen Typs zulassen, wobei auch verborgene Variablen erlaubt sind.

Empfehlungssysteme

Empfehlungssysteme werden immer mehr zu einem Hauptanwendungsgebiet des maschinellen Lernens. Viele Einzelhandelsunternehmen haben ein Interesse daran, das zukünftige Kundenverhalten auf der Basis früherer Verkaufsdaten vorherzusagen. Wir können die Daten mithilfe einer Matrix visualisieren, in der die Zeilen die Kunden und die Spalten die Artikel repräsentieren. Die Matrixelemente sind dann zum Beispiel Bestellmengen oder Kundenbewertungen. Typischerweise ist eine solche Matrix sehr groß und sehr dünn besetzt – denn natürlich haben die meisten Kunden nur eine sehr kleine Auswahl der lieferbaren Artikel gekauft. Obwohl die Matrix sehr groß ist, hat sie nur einen kleinen Rang. Das liegt daran, dass es viele Abhängigkeiten zwischen den Daten gibt. Menschen überlassen

es nicht dem Zufall, was sie kaufen. Wer zum Beispiel ein Baby hat, kauft ähnliche Dinge wie andere Eltern von Kleinkindern. Bestimmte Produkte werden sehr oft zusammen gekauft, andere so gut wie nie. Es ist diese Art der Regelhaftigkeit, verursacht durch eine kleine Zahl verborgener Faktoren, die für den niedrigen Rang der Matrix sorgt. Wenn wir uns in Kapitel 6 mit der Dimensionalitätsreduktion befassen, werden wir sehen, wie wir solche verborgenen Faktoren bzw. Abhängigkeiten aus den Daten extrahieren können.

3.7 Übungen

1. Angenommen, eine seltene Krankheit tritt nur bei einer von einer Million Personen auf. Nehmen wir weiter an, dass es einen Test gibt, der für eine Person, welche die Krankheit hat, mit einer Wahrscheinlichkeit von 99 Prozent ein positives Ergebnis liefert. Der Test ist allerdings nicht perfekt, und daher liefert er auch für eine von tausend gesunden Personen ein positives Ergebnis. Wie groß ist die Wahrscheinlichkeit, dass ein neuer, positiv getesteter Patient die Krankheit tatsächlich hat?

   Lösung: Wir wollen für die Krankheit und das Testergebnis die logischen Variablen d und t verwenden. Damit können wir die Aussagen der Aufgabenstellung wie folgt formulieren: $P(d=1) = 10^{-6}$, $P(t=1|d=1) = 0{,}99$, $P(t=1|d=0) = 10^{-3}$. Gefragt ist nach $P(d=1|t=1)$.

   Wir wenden den Satz von Bayes an:

$$\begin{aligned} P(d=1|t=1) &= \frac{P(t=1|d=1)P(d=1)}{P(t=1)} \\ &= \frac{P(t=1|d=1)P(d=1)}{P(t=1|d=1)P(d=1) + P(t=1|d=0)P(d=0)} \\ &= \frac{0{,}99 \cdot 10^{-6}}{0{,}99 \cdot 10^{-6} + 10^{-3} \cdot (1 - 10^{-6})} = 0{,}00098902\,. \end{aligned}$$

   Wenn wir also wissen, dass das Testergebnis positiv ist, dann erhöht sich für die getestete Person die Wahrscheinlichkeit, krank zu sein, von eins zu einer Million auf eins zu Tausend.

2. Bei einem Zweiklassenproblem ergibt sich das *Wahrscheinlichkeitsverhältnis* (engl. *likelihood ratio*) aus

   Wahrscheinlichkeitsverhältnis

$$\frac{p(\boldsymbol{x}|\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)}\,.$$

   Schreiben Sie die Diskriminanzfunktion als Ausdruck des Wahrscheinlichkeitsverhältnisses.

Lösung: Wir definieren eine Diskriminanzfunktion durch

$$g(x) = \frac{P(C_1|x)}{P(C_2|x)}$$

und wählen C_1 falls $g(x) > 0$, C_2 sonst. Die Diskriminanzfunktion kann als Produkt des Wahrscheinlichkeitsverhältnisses und dem Verhältnis der a-priori-Wahrscheinlichkeiten geschrieben werden:

$$g(x) = \frac{p(x|C_1)P(C_1)}{p(x|C_2)P(C_2)} .$$

Wenn die a-priori-Wahrscheinlichkeiten gleich sind, ist die Diskriminanzfunktion durch das Wahrscheinlichkeitsverhältnis gegeben.

3. Bei einem Zweiklassenproblem ist das *logarithmierte Chancenverhältnis* (log odds) definiert als

Log Odds

$$\log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{P(\mathcal{C}_2|\boldsymbol{x})} .$$

Schreiben Sie die Diskriminanzfunktion als Ausdruck der logarithmierten Chancenverhältnisse.

Lösung: Wir definieren eine Diskriminanzfunktion durch

$$g(x) = \log \frac{P(C_1|x)}{P(C_2|x)}$$

und wählen C_1 falls $g(x) > 0$, C_2 sonst. Log odds ist die Summe aus dem logarithmierten Wahrscheinlichkeitsverhältnis und dem Logarithmus des Verhältnisses der a-priori-Wahrscheinlichkeiten:

$$g(x) = \log \frac{p(x|C_1)}{p(x|C_2)} + \log \frac{P(C_1)}{P(C_2)} .$$

Wenn die a-priori-Wahrscheinlichkeiten gleich sind, ist die Diskriminanzfunktion das logarithmierte Wahrscheinlichkeitsverhältnis.

4. Bestimmen Sie für ein Problem mit zwei Klassen, zwei Aktionen und der gegebenen Verlustfunktion von $\lambda_{11} = \lambda_{22} = 0$, $\lambda_{12} = 10$, und $\lambda_{21} = 5$ die optimale Entscheidungsregel. Wie ändert sich die Regel, wenn wir als dritte Aktion die Ablehnung mit $\lambda = 1$ hinzufügen?

Lösung: Die Verlusttabelle sieht folgendermaßen aus:

Aktion	C_1	C_2
α_1: wähle C_1	0	10
α_2: wähle C_2	5	0

Berechnen wir nun die erwarteten Risiken der beiden Aktionen:

$$R(\alpha_1|x) = 0 \cdot P(C_1|x) + 10 \cdot P(C_2|x) = 10 \cdot (1 - P(C_1|x))$$
$$R(\alpha_2|x) = 5 \cdot P(C_1|x) + 0 \cdot P(C_2|x) = 5 \cdot P(C_1|x)\,.$$

Wir wählen α_1, wenn

$$\begin{aligned} R(\alpha_1|x) &< R(\alpha_2|x) \\ 10 \cdot (1 - P(C_1|x)) &< 5 \cdot P(C_1|x) \\ P(C_1|x) &> 2/3\,. \end{aligned}$$

Wenn die beiden Fehlklassifikationen die gleichen Kosten hätten, läge der Entscheidungsschwellwert bei $1/2$. Da jedoch die Kosten für die falsche Wahl von C_1 höher sind, wollen wir C_1 nur dann wählen, wenn wir wirklich sicher sind (siehe Abbildung 3.2, Teil a und b).

Wenn wir eine Option zum Ablehnen mit Kosten von 1 hinzufügen, haben wir die Verlusttabelle

Aktion	C_1	C_2
α_1: wähle C_1	0	10
α_2: wähle C_2	5	0
α_r: ablehnen	1	1

Berechnen wir nun die erwarteten Risiken der drei Aktionen:

$$R(\alpha_1|x) = 0 \cdot P(C_1|x) + 10 \cdot P(C_2|x) = 10 \cdot (1 - P(C_1|x))$$
$$R(\alpha_2|x) = 5 \cdot P(C_1|x) + 0 \cdot P(C_2|x) = 5 \cdot P(C_1|x)$$
$$R(\alpha_r|x) = 1\,.$$

Wir wählen α_1, falls $R(\alpha_1|x) < 1 \Rightarrow P(C_1|x) > 9/10$.

Wir wählen α_2, falls $R(\alpha_2|x) < 1 \Rightarrow P(C_1|x) < 1/5$, oder äquivalent, falls $P(C_2|x) > 4/5$.

In allen anderen Fällen, d. h. für $1/5 < P(C_1|x) < 9/10$ (siehe Abbildung 3.2, Teil c), erfolgt eine Ablehnung.

5. Gegeben sei eine Kaskade mit drei Niveaus, in der bei Ablehnung auf einem Niveau wie in Gleichung 3.10 das nächste Niveau verwendet wird. Wie können wir die λ in den verschiedenen Niveaus festlegen?

6. Jemand wirft eine ideale Münze und wenn das Ergebnis Kopf lautet, bekommen Sie nichts; andernfalls erhalten Sie 5 Euro. Wie hoch wäre Ihr Einsatz für dieses Spiel? Was wäre, wenn Sie statt der 5 Euro im Gewinnfall 500 Euro erhalten?

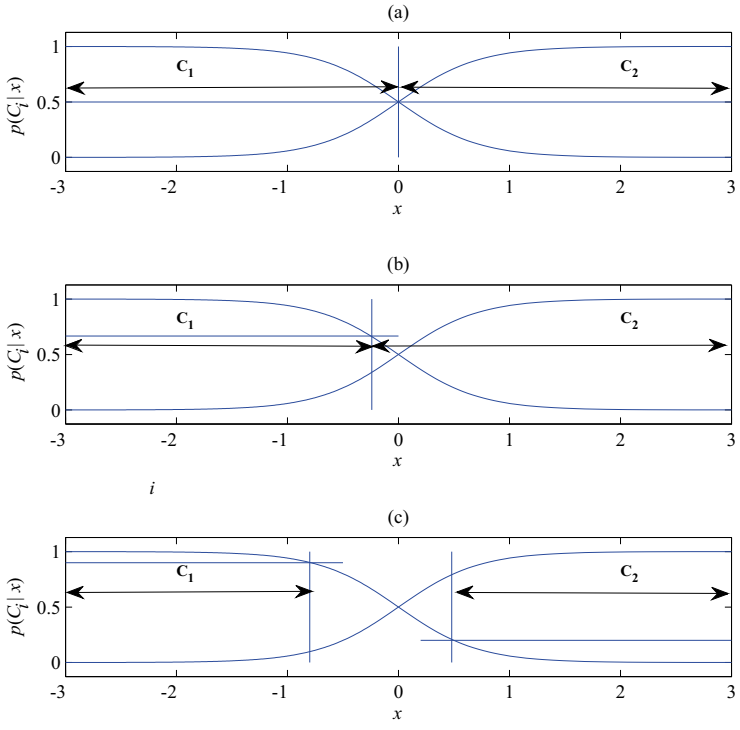


***Abb. 3.2*:** Die Grenze ändert sich, wenn sich die Verluste durch Fehlklassifikationen ändern. (a) Die Grenze liegt dort, wo die beiden a-posteriori-Wahrscheinlichkeiten gleich sind, wenn die beiden Fehlklassifikationen die gleichen Kosten haben. (b) Wenn die Verluste nicht symmetrisch sind, verschiebt sich die Klasse in Richtung der Klasse, die im Falle einer Fehlklassifikation ein höheres Risiko hat. (c) Wenn es die Option der Ablehnung gibt, ist die Umgebung der Grenze der Ablehnungsbereich.

7. Für die Transaktionen in einem Shop seien die folgenden Daten gegeben. Berechnen Sie die Support- und Konfidenzwerte von Milch → Bananen, Bananen → Milch, Milch → Schokolade und Schokolade → Milch.

Transaktion	gekaufte Artikel
1	Milch, Bananen, Schokolade
2	Milch, Schokolade
3	Milch, Bananen
4	Schokolade
5	Schokolade
6	Milch, Schokolade

Lösung:

Milch → Bananen:	Support=2/6, Konfidenz=2/4
Bananen → Milch:	Support=2/6, Konfidenz=2/2
Milch → Schokolade:	Support=3/6, Konfidenz=3/4
Schokolade → Milch:	Support=3/6, Konfidenz=3/5

Während nur die Hälfte der Kunden, die Milch kaufen, auch Bananen kaufen, kauft jeder, der Bananen kauft, auch Milch.

8. Verallgemeinern Sie die Formeln für die Konfidenz und den Support bei der Warenkorbanalyse, um auch k-Abhängigkeiten, also $P(Y|X_1, \ldots, X_k)$, berechnen zu können.

9. Zeigen Sie, dass die Konfidenz niemals wächst, wenn wir einen Artikel aus der Gruppe der nachrangigen Artikel in die der vorrangigen verschieben: Konfidenz$(ABC \to D) \geq$ Konfidenz$(AB \to CD)$

10. Angenommen, eine Warenkorbanalyse liefert zusammen mit jedem verkauften Artikel eine Zahl, die angibt, wie sehr der Kunde von dem Produkt begeistert ist – beispielsweise auf einer Skala von 0 bis 10. Wie lässt sich anhand dieser zusätzlichen Information berechnen, welches Produkt einem Kunden vorgeschlagen wird?

11. Schreiben Sie Beispiele für Transaktionsdaten auf, bei denen für die Regel $X \to Y$ gilt:

    (a) Sowohl der Support als auch die Konfidenz sind hoch.

    (b) Der Support ist hoch und die Konfidenz ist gering.

    (c) Der Support ist gering und die Konfidenz ist hoch.

    (d) Sowohl der Support als auch die Konfidenz sind gering.

3.8 Literaturangaben

Agrawal, R., H. Mannila, R. Srikant, H. Toivonen and A. Verkamo. 1996. „Fast Discovery of Association Rules.“ In *Advances in Knowledge Discovery and Data Mining,* ed. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, 307–328. Cambridge, MA: The MIT Press.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Li, J. 2006. „On Optimal Rule Discovery.“ *IEEE Transactions on Knowledge and Data Discovery* 18: 460–471.

Newman, J. R., ed. 1988. *The World of Mathematics.* Redmond, WA: Tempus.

Omiecinski, E. R. 2003. „Alternative Interest Measures for Mining Associations in Databases.“ *IEEE Transactions on Knowledge and Data Discovery* 15: 57–69.

Russell, S., and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach.* 3rd ed. New York: Prentice Hall.

Shafer, G., and J. Pearl, eds. 1990. *Readings in Uncertain Reasoning.* San Mateo, CA: Morgan Kaufmann.

Zhang, C., and S. Zhang. 2002. *Association Rule Mining: Models and Algorithms.* New York: Springer.

4 Parametrische Methoden

Nachdem wir diskutiert haben, wie man optimale Entscheidungen trifft, wenn bestehende Ungewissheit mit Hilfe von Wahrscheinlichkeiten modelliert wird, wollen wir nun untersuchen, wie wir diese Wahrscheinlichkeiten anhand eines Trainingsdatensatzes berechnen können. Wir beginnen mit dem parametrischen Ansatz für die Klassifikation und Regression. In späteren Kapiteln werden wir die semiparametrischen und nichtparametrischen Ansätze behandeln. Wir führen das Verzerrung/Varianz-Dilemma ein und stellen Modellauswahlmethoden vor, mit denen ein Kompromiss zwischen Modellkomplexität und dem empirischen Fehler gefunden werden soll.

4.1 Einführung

Eine Statistik ist ein beliebiger, anhand einer gegebenen Stichprobe berechneter Wert. Bei der statistischen Inferenz treffen wir Entscheidungen anhand der Informationen, welche uns eine Stichprobe liefert. Unser erster Ansatz ist ein parametrischer und wir nehmen an, dass die Stichprobe aus einer Verteilung gezogen wurde, die einem bekannten Modell folgt, also beispielsweise eine Gauß-Verteilung. Der Vorteil des parametrischen Ansatzes liegt darin, dass das Modell bis auf eine kleine Anzahl an Parametern – zum Beispiel Mittelwert oder Varianz – definiert ist, den sogenannten *ausreichenden Statistiken* der Verteilung. Sobald diese Parameter anhand der Stichprobe geschätzt wurden, ist die gesamte Verteilung bekannt. Wir schätzen die Parameter der Verteilung anhand der gegebenen Stichprobe, fügen diese Ergebnisse in das angenommene Modell ein und erhalten eine geschätzte Verteilung, die wir dann nutzen, um eine Entscheidung zu treffen. Die Methode, welche wir zur Schätzung der Parameter einer Verteilung nutzen, ist die sogenannte Maximum-Likelihood-Schätzung. Wir führen außerdem die Bayessche Schätzung ein, die wir in Kapitel 16 ausführlicher behandeln werden.

Wir beginnen mit der *Dichteschätzung*, welche die allgemeine Form der Schätzung von $p(x)$ ist. Wir nutzen diese für die *Klassifikation*, bei der die geschätzten Dichten den Klassendichten $p(x|\mathcal{C}_i)$ und $P(\mathcal{C}_i)$ entsprechen, um dann in der Lage zu sein, die a-posteriori-Wahrscheinlichkeit $P(\mathcal{C}_i|x)$ zu berechnen und unsere Entscheidung zu treffen. Danach diskutieren

wir die *Regression*, bei der die geschätzte Dichte $p(y|x)$ ist. In diesem Kapitel ist x eindimensional, und somit sind die Verteilungen univariat. Wir verallgemeinern dies für den multivariaten Fall in Kapitel 5.

4.2 Maximum-Likelihood-Schätzung

Gehen wir einmal von einer unabhängigen und identisch verteilten Stichprobe $\mathcal{X} = \{x^t\}_{t=1}^N$ aus. Wir nehmen an, dass x^t Instanzen sind, die nach einer bekannten Wahrscheinlichkeitsdichte $p(x|\theta)$, die bis auf die Parameter θ definiert ist, gezogen wurden.

$$x^t \sim p(x|\theta) \,.$$

LIKELIHOOD

Wir wollen das θ finden, bei dem x^t am wahrscheinlichsten aus $p(x|\theta)$ gezogen wird. Da x^t unabhängig sind, ist die *Likelihood*, also die zu erwartende Beobachtung, für die Stichprobe $\mathcal{X}$ bei gegebenem Parameter θ das Produkt aus den Likelihoods der individuellen Punkte:

$$l(\theta|\mathcal{X}) \equiv p(\mathcal{X}|\theta) = \prod_{t=1}^{N} p(x^t|\theta) \,. \tag{4.1}$$

MAXIMUM-LIKELIHOOD-SCHÄTZUNG

Bei der *Maximum-Likelihood-Schätzung* interessiert uns der Wert θ, durch den $\mathcal{X}$ mit größter Wahrscheinlichkeit gezogen wird. Wir suchen also nach dem θ, das die Likelihood der Stichprobe maximiert, was wir mit $l(\theta|\mathcal{X})$ bezeichnen. Wir können den Logarithmus der Likelihood maximieren, ohne den Wert zu verändern, an dem das Maximum auftritt. $\log(\cdot)$ konvertiert das Produkt in eine Summe und führt zu weiterer Vereinfachung des Rechenaufwands, falls gewisse Dichten vorausgesetzt werden,

LOG-LIKELIHOOD

beispielsweise das Vorhandensein von Exponenten. Die *Log-Likelihood* wird definiert als

$$\mathcal{L}(\theta|\mathcal{X}) \equiv \log l(\theta|\mathcal{X}) = \sum_{t=1}^{N} \log p(x^t|\theta) \,. \tag{4.2}$$

Betrachten wir nun einige Verteilungen, die in den Anwendungen auftreten, an denen wir interessiert sind. Wenn uns ein Zweiklassenproblem vorliegt, benutzen wir eine *Bernoulli-Verteilung*. Wenn es $K > 2$ Klassen gibt, handelt es sich um eine *Multinomialverteilung*. Die (*normale*) *Gauß-Verteilung* ist eine der am meisten benutzten für die Modellierung von klassenbezogenen Eingabedichten mit numerischer Eingabe. Für diese drei Verteilungen werden wir jeweils den Maximum-Likelihood-Schätzer (engl. *maximum likelihood estimator*, MLE) ihrer Parameter diskutieren.

4.2.1 Bernoulli-Verteilung

Bei einer Bernoulli-Verteilung gibt es zwei mögliche Ergebnisse: ein Ereignis tritt entweder ein oder nicht. Beispielsweise ist eine Instanz entweder ein positives Beispiel einer Klasse oder eben nicht. Mit der Wahrscheinlichkeit p tritt das Ereignis ein und die Bernoulli-Zufallsvariable X nimmt den Wert 1 an; das Nichteintreten des Ereignisses hat die Wahrscheinlichkeit $1-p$ und wird dadurch gekennzeichnet, dass X den Wert 0 bekommt. In geschriebener Form heißt das:

$$P(x) = p^x(1-p)^{1-x}, x \in \{0,1\} . \tag{4.3}$$

Erwartungswert und Varianz können wie folgt berechnet werden:

$$E[x] = \sum_x xp(x) = 1 \cdot p + 0 \cdot (1-p) = p ,$$
$$\mathrm{Var}(x) = \sum_x (x - E[X])^2 p(x) = p(1-p) .$$

p ist der einzige Parameter, und unter der Annahme einer unabhängigen, identisch verteilten Stichprobe $\mathcal{X} = \{x^t\}_{t=1}^N$ mit $x^t \in \{0,1\}$ wollen wir seinen Schätzer $\hat{p}$ berechnen. Die Log-Likelihood ergibt sich aus:

$$\begin{aligned}\mathcal{L}(p|\mathcal{X}) &= \log \prod_{t=1}^N p^{(x^t)}(1-p)^{(1-x^t)} \\ &= \sum_t x^t \log p + \left(N - \sum_t x^t\right) \log(1-p) .\end{aligned}$$

Den Wert für $\hat{p}$ zur Maximierung der Log-Likelihood findet man, indem man nach $d\mathcal{L}/dp = 0$ auflöst. Das ˆ (Zirkumflex) zeigt an, dass es sich um eine Schätzung handelt:

$$\hat{p} = \frac{\sum_t x^t}{N} . \tag{4.4}$$

Die Schätzung für p ist das Verhältnis zwischen der Häufigkeit des Eintretens des Ereignisses und der Anzahl der Experimente. Es sei erneut darauf hingewiesen, dass X eine Bernoulli-Verteilung mit p, $E[X] = p$ ist; somit ist, wie erwartet, der MLE des Mittelwertes der Durchschnitt der Stichprobe.

Man beachte, dass die Schätzung eine Funktion der Stichprobe und somit eine weitere Zufallsvariable ist; wir können von der Verteilung der $\hat{p}_i$ für verschiedene gegebene $\mathcal{X}_i$ sprechen, die nach dem gleichen $p(x)$ verteilt sind. Beispielsweise ist zu erwarten, dass die Varianz der Verteilung von $\hat{p}_i$ mit wachsendem N abnimmt. Je größer die Stichproben werden, umso ähnlicher werden sie (und folglich ihre Mittelwerte) einander.

4.2.2 Multinomiale Dichte

Wir betrachten nun die Verallgemeinerung der Bernoulli-Verteilung. Es gibt jetzt nicht nur zwei mögliche Zustände, sondern das Ergebnis eines Zufallsereignisses ist einer von K sich gegenseitig ausschließenden und erschöpfenden Zuständen, beispielsweise Klassen, von denen jede die Auftretenswahrscheinlichkeit p_i mit $\sum_{i=1}^{K} p_i = 1$ besitzt. $x_1, x_2, \ldots, x_K$ seien die Indikatorvariablen, wobei x_i gleich 1 ist, wenn das Ergebnis der Zustand i ist; andernfalls ist x_i gleich 0.

$$P(x_1, x_2, \ldots, x_K) = \prod_{i=1}^{K} p_i^{x_i} \tag{4.5}$$

Sagen wir nun, dass wir N unabhängige Experimente mit den Ergebnissen $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^{N}$ durchführen, wobei

$$x_i^t = \begin{cases} 1 \text{ falls Experiment } t \text{ den Zustand } i \text{ wählt}, \\ 0 \text{ andernfalls} \end{cases}$$

mit $\sum_i x_i^t = 1$. Der MLE von p_i ist

$$\hat{p}_i = \frac{\sum_t x_i^t}{N} . \tag{4.6}$$

Die Schätzung der Wahrscheinlichkeit des Zustands i ist das Verhältnis von Experimenten mit Zustand i als Ergebnis zur Gesamtzahl aller Experimente. Es gibt zwei Möglichkeiten, wie man auf dies kommen kann: Wenn x_i gleich 0/1 sind, dann kann man sich sie auch als K getrennte Bernoulli-Experimente vorstellen. Oder aber man schreibt explizit die Log-Likelihood und findet die p_i, wodurch sie maximiert wird (unter der Bedingung, dass $\sum_i p_i = 1$).

4.2.3 Gauß-Verteilung (Normalverteilung)

X unterliegt einer Gauß-Verteilung (auch als Normalverteilung bezeichnet) mit Mittelwert μ und Varianz σ^2, geschrieben als $\mathcal{N}(\mu, \sigma^2)$, wenn die zugehörige Dichtefunktion wie folgt aussieht:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty . \tag{4.7}$$

Für eine gegebene Stichprobe $\mathcal{X} = \{x^t\}_{t=1}^{N}$ mit $x^t \sim \mathcal{N}(\mu, \sigma^2)$ ergibt sich die Log-Likelihood

$$\mathcal{L}(\mu, \sigma | \mathcal{X}) = -\frac{N}{2} \log(2\pi) - N \log \sigma - \frac{\sum_t (x^t - \mu)^2}{2\sigma^2} .$$

Wir finden die MLE, indem wir die partiellen Ableitungen der Log-Likelihood bilden und sie gleich null setzen:

$$m = \frac{\sum_t x^t}{N}\,, \qquad (4.8)$$
$$s^2 = \frac{\sum_t (x^t - m)^2}{N}\,.$$

Wir halten uns an die allgemeine Vorgehensweise und nutzen Buchstaben des griechischen Alphabets für die Populationsparameter und römische Schriftzeichen für ihre Schätzungen anhand der Stichprobe. Manchmal wird auch ein Zirkumflex genutzt, um den Schätzer zu kennzeichnen, also zum Beispiel $\hat{\mu}$.

4.3 Bewertung eines Schätzers: Verzerrung und Varianz

Sei $\mathcal{X}$ eine Stichprobe aus einer Population, die bis auf die Parameter θ definiert ist, und sei $d = d(\mathcal{X})$ ein Schätzer für θ. Um die Qualität dieses Schätzers zu bewerten, können wir messen, wie stark er von θ abweicht, das heißt $(d(\mathcal{X}) - \theta)^2$. Da es sich aber um eine Zufallsvariable handelt (sie hängt von der Stichprobe ab), müssen wir den Durchschnitt über alle möglichen $\mathcal{X}$ errechnen und $r(d, \theta)$, den *mittleren quadratischen Fehler* des Schätzers d, der als

MITTLERER QUADRATISCHER FEHLER

$$b_\theta(d) = E[d(\mathcal{X})] - \theta \qquad (4.9)$$

definiert ist, in unsere Betrachtungen einbeziehen.

Die *Verzerrung* eines Schätzers ergibt sich aus

VERZERRUNG

$$b_\theta(d) = E[d(\mathcal{X})] - \theta\,. \qquad (4.10)$$

Wenn $b_\theta(d) = 0$ für alle Werte von θ, so sprechen wir von d als *verzerrungsfreiem Schätzer* von θ. Wenn zum Beispiel x^t aus einer Dichte mit dem Mittelwert μ gezogen wurde, so ist der Durchschnitt m der Stichprobe ein verzerrungsfreier Schätzer des Mittelwertes μ, denn

VERZERRUNGSFREIER SCHÄTZER

$$E[m] = E\left[\frac{\sum_t x^t}{N}\right] = \frac{1}{N}\sum_t E[x^t] = \frac{N\mu}{N} = \mu\,.$$

Das heißt, dass zwar bei einer speziellen Stichprobe m durchaus verschieden von μ sein kann, wenn wir aber viele solcher Stichproben $\mathcal{X}_i$ ziehen und viele $m_i = m(\mathcal{X}_i)$ schätzen,dann wird sich *deren* Durchschnitt

mit zunehmender Stichprobenzahl immer mehr an μ annähern. m ist außerdem ein *konsistenter* Schätzer, sprich $\mathrm{Var}(m) \to 0$ as $N \to \infty$.

$$\mathrm{Var}(m) = \mathrm{Var}\left(\frac{\sum_t x^t}{N}\right) = \frac{1}{N^2}\sum_t \mathrm{Var}(x^t) = \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}\,.$$

Mit wachsendem N, also der Anzahl an Punkten in der Stichprobe, weicht m immer weniger von μ ab. Überprüfen wir nun s^2, den MLE von σ^2:

$$\begin{aligned} s^2 &= \frac{\sum_t (x^t - m)^2}{N} = \frac{\sum_t (x^t)^2 - Nm^2}{N} \\ E[s^2] &= \frac{\sum_t E[(x^t)^2] - N \cdot E[m^2]}{N}\,. \end{aligned}$$

Gesetzt dem Fall, dass $\mathrm{Var}(X) = E[X^2] - E[X]^2$, erhalten wir $E[X^2] = \mathrm{Var}(X) + E[X]^2$, und wir können schreiben

$$E[(x^t)^2] = \sigma^2 + \mu^2 \text{ sowie } E[m^2] = \sigma^2/N + \mu^2\,.$$

Wenn wir dies einsetzen, erhalten wir

$$E[s^2] = \frac{N(\sigma^2 + \mu^2) - N(\sigma^2/N + \mu^2)}{N} = \left(\frac{N-1}{N}\right)\sigma^2 \neq \sigma^2\,,$$

wodurch nachgewiesen ist, dass s^2 ein verzerrter Schätzer von σ^2 ist. $(N/(N-1))s^2$ ist ein verzerrungsfreier Schätzer. Wenn N jedoch sehr groß ist, kann der Unterschied vernachlässigt werden. Dies ist ein Beispiel für einen *asymptotischen verzerrungsfreien Schätzer*, dessen Verzerrung gegen Null geht, wenn N gegen unendlich geht.

Der mittlere quadratische Fehler kann wie folgt umformuliert werden (d ist die Kurzform für $d(\mathcal{X})$):

$$\begin{aligned} r(d,\theta) &= E\left[(d-\theta)^2\right] \\ &= E\left[(d - E[d] + E[d] - \theta)^2\right] \\ &= E\left[(d-E[d])^2 + (E[d]-\theta)^2 + 2(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + E\left[(E[d]-\theta)^2\right] + 2E\left[(E[d]-\theta)(d-E[d])\right] \\ &= E\left[(d-E[d])^2\right] + (E[d]-\theta)^2 + 2(E[d]-\theta)E[d-E[d]] \\ &= \underbrace{E\left[(d-E[d])^2\right]}_{Varianz} + \underbrace{(E[d]-\theta)^2}_{Verzerrung^2}\,. \end{aligned} \tag{4.11}$$

Die beiden Gleichungen ergeben sich, weil $E[d]$ eine Konstante ist und damit $E[d] - \theta$ ebenfalls eine Konstante ist, und weil $E[d - E[d]] = E[d] - E[d] = 0$. In Gleichung 4.11 ist der erste Term die *Varianz*, welche

VARIANZ

misst, wie stark d_i im Durchschnitt um den Erwartungswert herum variiert (von einem Datensatz zum nächsten). Der zweite Term ist die *Verzerrung*, die misst, wie stark der Erwartungswert vom korrekten Wert θ abweicht (Abbildung 4.1). Wir schreiben den Fehler dann als Summe dieser beiden Terme, also der Varianz und des Quadrates der Verzerrung:

$$r(d,\theta) = \mathrm{Var}(d) + (b_\theta(d))^2 \,. \tag{4.12}$$

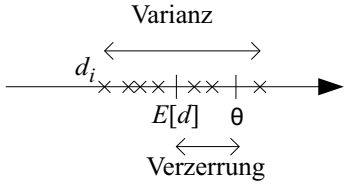


Abb. 4.1: θ ist der zu schätzende Parameter. d_i sind verschiedene Schätzungen (gekennzeichnet durch „×“) über verschiedene Stichproben. Die Verzerrung ist die Differenz zwischen dem Erwartungswert für d und θ. Die Varianz ergibt sich daraus, wie stark die d_i rings um den Erwartungswert verteilt liegen. Wir möchten für beide möglichst niedrige Werte erhalten.

4.4 Der Bayessche Schätzer

In manchen Fällen haben wir (oder Experten für den jeweiligen Anwendungsfall) möglicherweise schon Informationen zum möglichen Wertebereich für einen bestimmten Parameter θ, *bevor* wir uns die Stichprobe ansehen. Derlei Information ist durchaus nützlich und sollte einbezogen werden, ganz besonders dann, wenn nur eine kleine Stichprobe vorliegt. Die vorher verfügbaren Informationen geben uns aber keine Auskunft über den exakten Parameterwert (sonst bräuchten wir die Stichprobe nicht), und diese Ungewissheit modellieren wir, indem wir θ als Zufallsvariable betrachten und eine a-priori-Dichte $p(\theta)$ für sie definieren. Nehmen wir zum Beispiel an, uns wurde mitgeteilt, dass θ annähernd normalverteilt ist und mit 90-prozentiger Sicherheit zwischen 5 und 9 liegt, und zwar symmetrisch um 7 herum. Dann können wir $p(\theta)$ als normalverteilt mit dem Mittelwert 7 notieren, und weil

$$\begin{aligned} P\{-1{,}64 < \frac{\theta-\mu}{\sigma} < 1{,}64\} &= 0{,}9\,, \\ P\{\mu - 1{,}64\sigma < \theta < \mu + 1{,}64\sigma\} &= 0{,}9 \end{aligned}$$

nehmen wir $1{,}64\sigma = 2$ und nutzen $\sigma = 2/1{,}64$. Wir können daher annehmen, dass $p(\theta) \sim \mathcal{N}(7, (2/1{,}64)^2)$.

Die *a-priori-Dichte* $p(\theta)$ gibt uns Auskunft über die wahrscheinlichen Werte von θ, *bevor* wir die Stichprobe betrachten. Durch Anwendung des

A-PRIORI-DICHTE

Satzes von Bayes kombinieren wir dies mit den Informationen, die wir den Stichprobendaten entnehmen können, sprich der Likelihood-Dichte $p(\mathcal{X}|\theta)$ und erhalten die *a-posteriori-Dichte* von θ, die uns die wahrscheinlichen Werte für θ aufzeigt, *nachdem* wir die Stichprobe betrachtet haben:

A-POSTERIORI-DICHTE

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\theta)p(\theta)}{\int p(\mathcal{X}|\theta')p(\theta')d\theta'} . \tag{4.13}$$

Um die Dichte bei x zu schätzen, haben wir

$$\begin{aligned} p(x|\mathcal{X}) &= \int p(x,\theta|\mathcal{X})d\theta , \\ &= \int p(x|\theta,\mathcal{X})p(\theta|\mathcal{X})d\theta , \\ &= \int p(x|\theta)p(\theta|\mathcal{X})d\theta . \end{aligned}$$

Es gilt $p(x|\theta,\mathcal{X}) = p(x|\theta)$, denn sobald wir θ kennen, also die ausreichenden Statistiken, wissen wir alles über die Verteilung. Somit berechnen wir einen Durchschnitt über alle Vorhersagen unter Zuhilfenahme aller Werte von θ, gewichtet durch ihre Wahrscheinlichkeiten. Wenn wir eine Vorhersage der Form $y = g(x|\theta)$ treffen, wie bei der Regression, so erhalten wir

$$y = \int g(x|\theta)p(\theta|\mathcal{X})d\theta .$$

Die Berechnung der Integrale kann sich als durchaus schwierig herausstellen, außer in den Fällen, bei denen die a-posteriori-Wahrscheinlichkeit von geeigneter Form ist. Ist die vollständige Integration nicht möglich, reduzieren wir das Ganze auf einen einzigen Punkt. Wenn wir davon ausgehen können, dass $p(\theta|\mathcal{X})$ einen schmalen Scheitelpunkt im Bereich seines häufigsten Wertes hat, dann wird die Berechnung durch eine *Maximum-a-posteriori-Schätzung* (MAP) erleichtert:

MAXIMUM-A-POSTERIORI-SCHÄTZUNG

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathcal{X}) , \tag{4.14}$$

wodurch die komplette Dichte durch einen einzigen Punkt ersetzt und somit das Integral entfernt wird; wir erhalten:

$$\begin{aligned} p(x|\mathcal{X}) &= p(x|\theta_{MAP}) , \\ y_{MAP} &= g(x|\theta_{MAP}) . \end{aligned}$$

Wenn uns a priori keine Gründe vorliegen, aufgrund derer wir bestimmte Werte für θ vorziehen sollten, so ist die a-priori-Dichte gleich der Dichte

einer Gleichverteilung. Das hat zur Folge, dass die a-posteriori-Dichte die gleiche Form wie die Likelihood $p(\mathcal{X}|\theta)$ besitzt, weshalb die MAP-Schätzung gleich der Maximum-Likelihood-Schätzung (Abschnitt 4.2) ist, bei der wir

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{X}|\theta) \tag{4.15}$$

vorliegen haben.

Eine weitere Möglichkeit bietet der *Bayessche Schätzer*, der als der Erwartungswert der a-posteriori-Dichte definiert ist:

BAYESSCHER SCHÄTZER

$$\theta_{Bayes} = E[\theta|\mathcal{X}] = \int \theta p(\theta|\mathcal{X}) d\theta \,. \tag{4.16}$$

Der Grund dafür, dass der Erwartungswert gewählt wird, liegt darin, dass die beste Schätzung einer Zufallsvariable ihr Mittelwert ist. Nehmen wir an, dass θ die Variable ist, welche wir mit $E[\theta] = \mu$ vorhersagen wollen. Wenn eine Konstante c unsere Schätzung für θ ist, dann gilt nachweislich:

$$\begin{aligned} E[(\theta - c)^2] &= E[(\theta - \mu + \mu - c)^2] \\ &= E[(\theta - \mu)^2] + (\mu - c)^2 \,. \end{aligned} \tag{4.17}$$

Dabei wird das Minimum erreicht, wenn c als μ gewählt wird. Im Falle einer normalverteilten Dichte ist der häufigste Wert der Erwartungswert, und somit gilt, wenn $p(\theta|\mathcal{X})$ normalverteilt ist, $\theta_{Bayes} = \theta_{MAP}$.

Als Beispiel nehmen wir einmal an, dass $x^t \sim \mathcal{N}(\theta, \sigma_0^2)$ und $\theta \sim \mathcal{N}(\mu, \sigma^2)$, wobei μ, σ und σ_0^2 bekannt sind:

$$\begin{aligned} p(\mathcal{X}|\theta) &= \frac{1}{(2\pi)^{N/2}\sigma_0^N} \exp\left[-\frac{\sum_t (x^t - \theta)^2}{2\sigma_0^2}\right] \,, \\ p(\theta) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(\theta - \mu)^2}{2\sigma^2}\right] \,. \end{aligned}$$

Es kann gezeigt werden, dass $p(\theta|\mathcal{X})$ normalverteilt ist mit

$$E[\theta|\mathcal{X}] = \frac{N/\sigma_0^2}{N/\sigma_0^2 + 1/\sigma^2} m + \frac{1/\sigma^2}{N/\sigma_0^2 + 1/\sigma^2} \mu \,. \tag{4.18}$$

Somit ist der Bayessche Schätzer ein gewichteter Durchschnittswert des a-priori-Mittelwertes μ_0 und des Mittelwertes m der Stichprobe, wobei die Gewichte umgekehrt proportional zu ihren Varianzen sind. Mit wachsender Stichprobe N nähert sich der Bayessche Schätzer dem Stichprobendurchschnitt an, da mehr Informationen aus der Stichprobe genutzt werden können. Wenn σ_0^2 klein ist, d. h., wenn es anfangs nur wenig Ungewissheit hinsichtlich des korrekten Wertes für θ gibt, oder wenn N klein ist, dann hat unsere a-priori-Vermutung μ_0 einen stärkeren Effekt.

Man beachte, dass sowohl die MAP-Schätzung als auch der Bayessche Schätzer die komplette a-posteriori-Dichte auf einen einzigen Punkt reduzieren und damit ein Informationsverlust einhergeht, es sei denn, die a-posteriori-Verteilung ist unimodal und weist einen spitzen Scheitel nahe dieser Punkte auf. Mit zunehmend niedrigeren Berechnungskosten bietet sich eine Möglichkeit in der Benutzung eines Monte-Carlo-Ansatzes, der Stichproben anhand der a-posteriori-Dichte generiert (Andrieu et al. 2003). Außerdem gibt es Approximationsmethoden, die zur Bewertung des kompletten Integrals genutzt werden können. Wir werden die Bayessche Schätzung in Kapitel 16 ausführlicher diskutieren.

4.5 Parametrische Klassifikation

In Kapitel 3 haben wir gesehen, dass wir unter Verwendung des Satzes von Bayes die a-posteriori-Wahrscheinlichkeit der Klasse $\mathcal{C}_i$ als

$$P(\mathcal{C}_i|x) = \frac{p(x|\mathcal{C}_i)P(\mathcal{C}_i)}{p(x)} = \frac{p(x|\mathcal{C}_i)P(\mathcal{C}_i)}{\sum_{k=1}^{K} p(x|\mathcal{C}_k)P(\mathcal{C}_k)} \tag{4.19}$$

schreiben können und die Diskriminanzfunktion

$$g_i(x) = p(x|\mathcal{C}_i)P(\mathcal{C}_i)$$

oder äquivalent dazu

$$g_i(x) = \log p(x|\mathcal{C}_i) + \log P(\mathcal{C}_i) \tag{4.20}$$

nutzen können.

Wenn wir davon ausgehen, dass $p(x|\mathcal{C}_i)$ einer Gauß-Verteilung unterliegt, also

$$p(x|\mathcal{C}_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right] , \tag{4.21}$$

dann wird Gleichung 4.20 zu

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log \sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(\mathcal{C}_i) . \tag{4.22}$$

Betrachten wir das Ganze anhand eines Beispiels. Nehmen wir an, wir leiten ein Autohaus, welches K verschiedene Autos verkauft. Gehen wir aus Gründen der Einfachheit weiterhin davon aus, dass der einzige Faktor, der die Kaufentscheidung eines Kunden beeinflusst, dessen Jahreseinkommen ist, das wir mit x bezeichnen. Dann ist $P(\mathcal{C}_i)$ der Anteil an Kunden, die einen Wagen vom Typ i kaufen. Wenn die Verteilung der jährlichen Einkommen dieser Kunden mit einer Gauß-Verteilung approximiert werden

kann, so kann $p(x|\mathcal{C}_i)$, also die Wahrscheinlichkeit dafür, dass ein Kunde, der einen Wagen vom Typ i gekauft hat, ein Einkommen x aufweist, als $\mathcal{N}(\mu_i, \sigma_i^2)$ gesetzt werden, wobei μ_i das mittlere Einkommen solcher Kunden ist und σ_i^2 für deren Einkommensvarianz steht.

Wenn uns $P(\mathcal{C}_i)$ und $p(x|\mathcal{C}_i)$ nicht bekannt sind, so schätzen wir sie anhand einer Stichprobe und ergänzen die geschätzten Werte, um die Schätzung für die Diskriminanzfunktion zu erhalten. Uns liegt eine Stichprobe vor mit

$$\mathcal{X} = \{x^t, \boldsymbol{r}^t\}_{t=1}^N \,, \tag{4.23}$$

wobei $x \in \mathbb{R}$ eindimensional ist und $\boldsymbol{r} \in \{0,1\}^K$, so dass

$$r_i^t = \begin{cases} 1 \text{ falls } \boldsymbol{x}^t \in \mathcal{C}_i \,, \\ 0 \text{ falls } \boldsymbol{x}^t \in \mathcal{C}_k, k \neq i \,. \end{cases} \tag{4.24}$$

Für jede Klasse berechnen sich die Schätzungen für Mittelwert und Varianz (basierend auf Gleichung 4.8) durch

$$m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t} \,, \tag{4.25}$$

$$s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t} \tag{4.26}$$

und die Schätzungen für die a-priori-Werte (nach Gleichung 4.6) ergeben sich aus

$$\hat{P}(\mathcal{C}_i) = \frac{\sum_t r_i^t}{N} \,. \tag{4.27}$$

Setzen wir nun diese Werte in Gleichung 4.22 ein, so erhalten wir

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log s_i - \frac{(x - m_i)^2}{2s_i^2} + \log \hat{P}(\mathcal{C}_i) \,. \tag{4.28}$$

Der erste Term ist eine Konstante und kann weggelassen werden, da ihn alle $g_i(x)$ gemein haben. Wenn die a-priori-Werte gleich sind, kann der letzte Term ebenfalls weggestrichen werden. Wenn wir weiterhin annehmen, dass die Varianzen identisch sind, so können wir notieren

$$g_i(x) = -(x - m_i)^2 \tag{4.29}$$

und somit weisen wir x der Klasse mit dem naheliegendsten Mittelwert zu:

$$\text{Wähle } \mathcal{C}_i, \text{ falls } |x - m_i| = \min_k |x - m_k| \,.$$

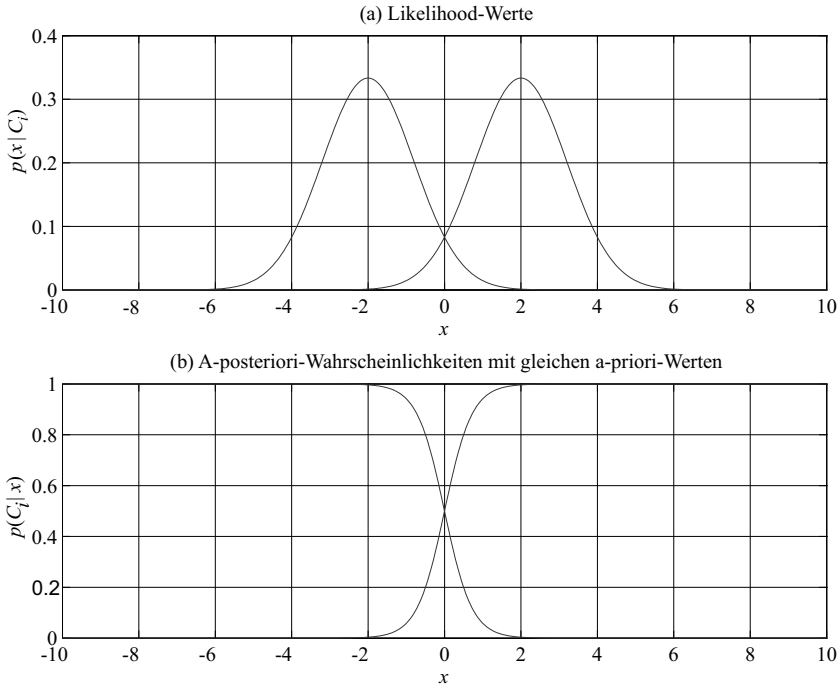


***Abb. 4.2*:** (a) Likelihood-Funktionen und (b) a-posteriori-Werte mit gleichen a-priori-Werten für zwei Klassen bei eindimensionaler Eingabe. Die Varianzen sind gleichwertig und die a-posteriori-Werte schneiden sich an einem Punkt, welcher den Schwellwert für die Entscheidung repräsentiert.

Bei zwei benachbarten Klassen ist der Mittelpunkt zwischen den zwei Mittelwerten der Schwellwert für die Entscheidung (siehe Abbildung 4.2).

$$\begin{aligned} g_1(x) &= g_2(x)\,, \\ (x-m_1)^2 &= (x-m_2)^2\,, \\ x &= \frac{m_1+m_2}{2}\,. \end{aligned}$$

Bei unterschiedlichen Varianzen existieren zwei Schwellwerte (siehe Abbildung 4.3), die sehr einfach zu berechnen sind (Übung 4). Wenn die a-priori-Werte sich unterscheiden, dann bewirkt dies eine Verschiebung des Entscheidungsschwellwertes in Richtung des Mittelwertes der weniger wahrscheinlichen Klasse.

Hier benutzen wir die Maximum-Likelihood-Schätzer für die Parameter. Wenn wir jedoch vorab schon Informationen über sie haben – beispielsweise für die Mittelwerte – so nutzen wir den Bayesschen Schätzer von $p(x|\mathcal{C}_i)$ mit a-priori-Werten für μ_i.

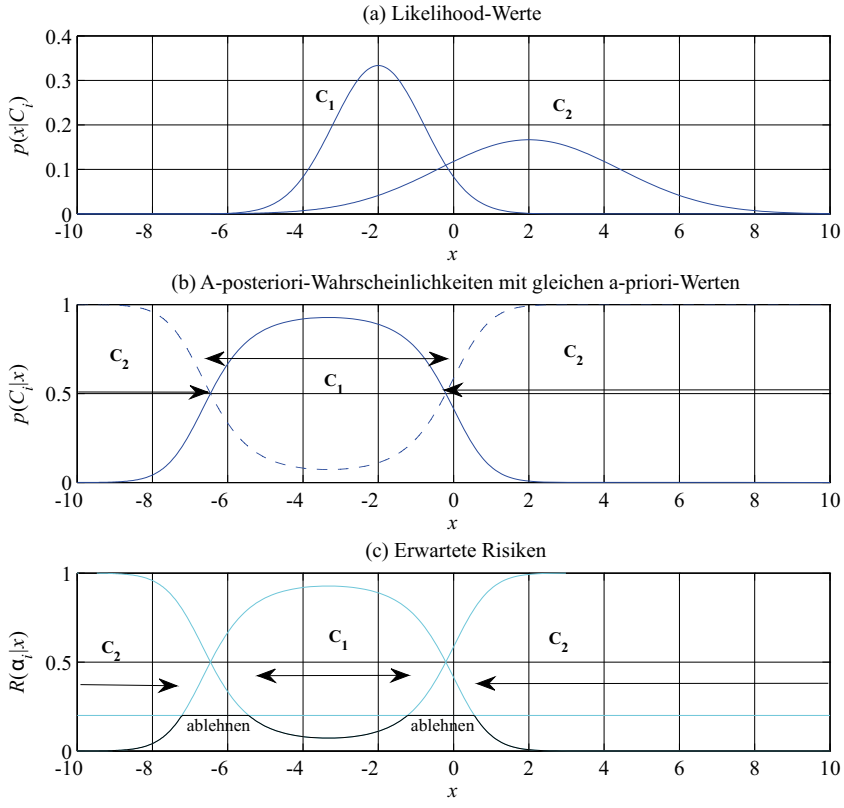


***Abb. 4.3*:** (a) Likelihood-Funktionen und (b) a-posteriori-Werte mit gleichen a-priori-Werten für zwei Klassen bei eindimensionaler Eingabe. Die Varianzen sind ungleich und die a-posteriori-Kurven schneiden sich in zwei Punkten. Teil (c) zeigt die erwarteten Risiken für die beiden Klassen, wobei mit $\lambda = 0{,}2$ abgelehnt wird (Abschnitt 3.3).

Ein wichtiger Hinweis ist hier vonnöten. Wenn x kontinuierlich ist, sollten wir nicht sofort auf die Gaußschen Dichten für $p(x|\mathcal{C}_i)$ zurückgreifen. Der Klassifikationsalgorithmus, sprich die Schwellwertpunkte, wird versagen, wenn die Dichten keiner Gauß-Verteilung unterliegen. In der Statisiklite-ratur existieren Tests zur Überprüfung auf Normalität, und ein solcher Test sollte angewendet werden, bevor Normalität vorausgesetzt wird. Im Falle von eindimensionalen Daten ist der einfachste Test der, das Histogramm abzubilden und visuell zu prüfen, ob die Dichte glockenförmig, also unimodal und symmetrisch um das Zentrum herum verteilt ist.

Dies ist der sogenannte *Likelihood-Ansatz* zur Klassifikation, bei dem wir Daten nutzen, um Dichten zu schätzen, den Satz von Bayes zur Berechnung der a-posteriori-Dichten einsetzen und dann die Diskriminante erhalten. In folgenden Kapiteln werden wir den *diskriminantenbasierten*

Ansatz diskutieren, bei dem wir die Schätzung der Dichten umgehen und direkt die Diskriminanten schätzen.

4.6 Regression

Bei der Regression besteht unser Ziel darin, die numerische Ausgabe, die als *abhängige Variable* bezeichnet wird, als Funktion der Eingabe, der sogenannten *unabhängigen Variable*, zu schreiben. Wir gehen davon aus, dass die numerische Ausgabe die Summe einer deterministischen Funktion der Eingabe und zufälligen Rauschens ist:

$$r = f(x) + \epsilon \,,$$

wobei $f(x)$ die unbekannte Funktion bezeichnet, welche wir anhand unseres Schätzers $g(x|\theta)$, der bis auf die Parametermenge θ definiert ist, approximieren wollen. Wenn wir annehmen, dass ϵ einer Gauß-Verteilung mit Nullmittelwert und konstanter Varianz σ^2 entspricht, also $\epsilon \sim \mathcal{N}(0, \sigma^2)$, und wenn wir unseren Schätzer $g(\cdot)$ anstelle der unbekannten Funktion $f(\cdot)$ einsetzen, so erhalten wir (Abbildung 4.4)

$$p(r|x) \sim \mathcal{N}(g(x|\theta), \sigma^2) \,. \tag{4.30}$$

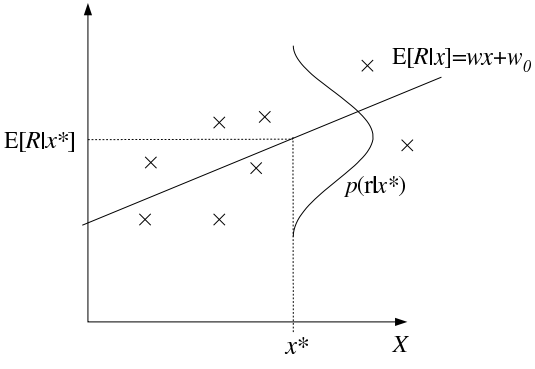


Abb. 4.4: Bei der Regression geht man davon aus, dass dem Modell Gaußsches Rauschen mit Nullmittelwert zugefügt wurde; hier handelt es sich um ein lineares Modell.

Erneut nutzen wir die Maximum-Likelihood-Methode, um die Parameter θ zu bestimmen. Die Tupel (x^t, r^t) im Trainingsdatensatz werden nach einer unbekannten gemeinsamen Wahrscheinlichkeitsdichte $p(x, r)$ gezogen, was wir in der Form

$$p(x, r) = p(r|x)p(x)$$

notieren können.

$p(r|x)$ ist die Wahrscheinlichkeit der Ausgabe bei vorliegender Eingabe, und $p(x)$ ist die Eingabedichte. Gegeben sei eine unabhängige, identisch

verteilte Stichprobe $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$; die Log-Likelihood ergibt sich dann aus

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} p(x^t, r^t) \\ &= \log \prod_{t=1}^{N} p(r^t|x^t) + \log \prod_{t=1}^{N} p(x^t) \,.\end{aligned}$$

Wir können den zweiten Term ignorieren, da er nicht von unserem Schätzer abhängt, und somit erhalten wir

$$\begin{aligned}\mathcal{L}(\theta|\mathcal{X}) &= \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{[r^t - g(x^t|\theta)]^2}{2\sigma^2}\right] \\ &= \log \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^N \exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2\right] \\ &= -N\log(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2 \,.\end{aligned} \tag{4.31}$$

Der erste Term ist unabhängig von den Parametern θ und kann weggelassen werden; das gleiche gilt für den Faktor $1/\sigma^2$. Die Maximierung des Ganzen ist äquivalent zur Minimierung von

$$E(\theta|\mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2 \,, \tag{4.32}$$

was die am häufigsten verwendete Fehlerfunktion ist. Die Werte für θ, welche diese Fehlerfunktion minimieren, nennt man auch Schätzwerte nach der Methode der *kleinsten Quadrate*. Diese Transformation wird in der Statistik häufig vorgenomme. Wenn die Likelihood l Exponenten enthält, so maximieren wir nicht l, sondern definieren stattdessen eine *Fehlerfunktion* $E = -\log l$ und minimieren diese.

Methode der kleinsten Quadrate

Bei der *linearen Regression* arbeiten wir mit einem linearen Modell

lineare Regression

$$g(x^t|w_1, w_0) = w_1 x^t + w_0 \,,$$

und wenn wir die Ableitung der Summe der quadrierten Fehler herbeinehmen (Gleichung 4.32), dann erhalten wir zwei Gleichungen mit zwei Unbekannten,

$$\begin{aligned}\sum_t r^t &= N w_0 + w_1 \sum_t x^t \,, \\ \sum_t r^t x^t &= w_0 \sum_t x_t + w_1 \sum_t (x^t)^2 \,.\end{aligned}$$

Diese können wir mittels Vektor- und Matrixnotation als $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ notieren mit

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix},$$

was nach $\boldsymbol{w} = \mathbf{A}^{-1}\boldsymbol{y}$ aufgelöst werden kann.

POLYNOMIALE REGRESSION

Beim allgemeinen Fall der *polynomialen Regression* liegt ein polynomiales Modell k-ten Grades vor,

$$g(x^t|w_k, \ldots, w_2, w_1, w_0) = w_k(x^t)^k + \cdots + w_2(x^t)^2 + w_1 x^t + w_0 .$$

Das Modell ist nach wie vor linear in den Parametern, und wenn wir die Ableitungen bilden, erhalten wir $k+1$ Gleichungen mit $k+1$ Unbekannten, die in Vektor- oder Matrixnotation in der Form $\mathbf{A}\boldsymbol{w} = \boldsymbol{y}$ geschrieben werden können:

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t & \sum_t (x^t)^2 & \cdots & \sum_t (x^t)^k \\ \sum_t x^t & \sum_t (x^t)^2 & \sum_t (x^t)^3 & \cdots & \sum_t (x^t)^{k+1} \\ \vdots & & & & \\ \sum_t (x^t)^k & \sum_t (x^t)^{k+1} & \sum_t (x^t)^{k+2} & \cdots & \sum_t (x^t)^{2k} \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \\ \sum_t r^t (x^t)^2 \\ \vdots \\ \sum_t r^t (x^t)^k \end{bmatrix}.$$

Wir können notieren: $\mathbf{A} = \mathbf{D}^T\mathbf{D}$ und $\boldsymbol{y} = \mathbf{D}^T\boldsymbol{r}$, mit

$$\mathbf{D} = \begin{bmatrix} 1 & x^1 & (x^1)^2 & \cdots & (x^1)^k \\ 1 & x^2 & (x^2)^2 & \cdots & (x^2)^k \\ \vdots & & & & \\ 1 & x^N & (x^N)^2 & \cdots & (x^N)^k \end{bmatrix}, \quad \boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}.$$

Aufgelöst nach den Parametern erhalten wir dann

$$\boldsymbol{w} = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\boldsymbol{r} . \tag{4.33}$$

Die Maximierung der Likelihood unter der Annahme eines gaußverteilten Fehlers entspricht der Minimierung der Summe der quadrierten Fehler.

RELATIVER QUADRATISCHER FEHLER

Eine weitere Meßgröße ist der *relative quadratische Fehler* (engl. *relative square error*, RSE):

$$E_{RSE} = \frac{\sum_t [r^t - g(x^t|\theta)]^2}{\sum_t (r^t - \overline{r})^2} . \tag{4.34}$$

Wenn E_{RSE} nahe 1 ist, so ist unsere Vorhersage genauso gut wie eine Vorhersage basierend auf dem Durchschnitt. Wenn sich der Wert 0 nähert, verbessert sich unsere Modellanpassung. Dass E_{RSE} nahe 1 ist, bedeutet, dass die Verwendung eines Modells, welches auf der Eingabe x basiert, nicht besser ist als ein Modell auf der Basis des Mittelwerts, der unser Schätzer wäre, wenn es kein x gäbe; läge E_{RSE} nahe 0, wäre die Eingabe x hilfreich.

Ein Maß, mit dem die Güte der Anpassung durch Regression bewertet werden kann, ist das *Bestimmtheitsmaß.* Dieses ist definiert als

BESTIMMTHEITSMASS

$$R^2 = 1 - E_{RSE} \,,$$

und damit eine Regression als sinnvoll angesehen werden kann, fordern wir, dass R^2 nahe 1 ist.

Es sei noch einmal daran erinnert, dass wir für die bestmögliche Generalisierung die Komplexität unseres Lernmodells an die Komplexität der Daten anpassen sollten. Bei der Polynomregression entspricht der Komplexitätsparameter dem Grad des angepassten Polynoms, und daher liegt es an uns, einen Weg zu finden, mit dem der beste Grad ausgewählt werden kann, der den Generalisierungsfehler minimiert, der also die Komplexität des Modells bestmöglich an die Komplexität der den Daten zu Grunde liegenden Funktion anpasst.

4.7 Anpassung der Modellkomplexität: Das Verzerrung/Varianz-Dilemma

Gehen wir von einer Stichprobe $\mathcal{X} = \{x^t, r^t\}$ aus, welche nach einer unbekannten gemeinsamen Wahrscheinlichkeitsdichte $p(x, r)$ gezogen wurde. Mit Hilfe dieser Stichprobe konstruieren wir unsere Schätzung $g(\cdot)$. Der erwartete quadratische Fehler (über die gemeinsame Dichte) bei x kann wie folgt notiert werden (Gleichung 4.17):

$$E[(r - g(x))^2|x] = \underbrace{E[(r - E[r|x])^2|x]}_{Rauschen} + \underbrace{(E[r|x] - g(x))^2}_{quadratischer\ Fehler} \,. \tag{4.35}$$

Der erste Term zur Rechten ist die Varianz von r bei gegebenem x; er ist nicht abhängig von $g(\cdot)$ oder $\mathcal{X}$ und entspricht der Varianz σ^2 des hinzugefügten Rauschens. Hierbei handelt es sich um den Anteil des Fehlers, der sich nie entfernen lässt, egal welchen Schätzer wir nutzen. Der zweite Term quantifiziert, wie stark $g(x)$ von der Regressionsfunktion $E[r|x]$ abweicht. Dies hängt wiederum vom Schätzer und dem Datensatz ab. Es kann passieren, dass $g(x)$ für eine Stichprobe eine sehr gute Anpassung darstellt, wohingegen es für eine andere Probe eine eher schlechte Anpassung bietet. Um zu bewerten wie gut ein Schätzer $g(\cdot)$ ist,

bilden wir den Durchschnitt seines quadratischen Fehlers für verschiedene, mögliche Datensätze.

Der Erwartungswert (Durchschnitt über Stichproben $\mathcal{X}$, welche alle von der Größe N sind und derselben gemeinsamen Verteilung $p(r,x)$ gehorchen) ist (man nutze Gleichung 4.11):

$$\begin{aligned} &E_{\mathcal{X}}[(E[r|x]-g(x))^2|x] \\ &\quad = \underbrace{(E[r|x]-E_{\mathcal{X}}[g(x)])^2}_{Verzerrung} + \underbrace{E_{\mathcal{X}}[(g(x)-E_{\mathcal{X}}[g(x)])^2]}_{Varianz} . \end{aligned} \tag{4.36}$$

Wie wir bereits besprochen haben, misst die Verzerrung, wie stark $g(x)$ vom Zielwert abweicht, ohne dabei den Effekt variierender Stichproben einzubeziehen, und die Varianz gibt an, wie stark $g(x)$ bei variierender Stichprobe um den Erwartungswert $E[g(x)]$ herum fluktuieren. Wir wollen, dass beide Werte klein sind.

Betrachten wir nun ein didaktisches Beispiel. Um die Verzerrung und die Varianz zu schätzen, generieren wir eine Menge an Datensätzen $\mathcal{X}_i = \{x_i^t, r_i^t\}, i = 1,\ldots,M$ anhand einer bekannten Funktion $f(\cdot)$ mit zugefügtem Rauschen, nutzen jeden Datensatz, um einen Schätzer $g_i(\cdot)$ zu formen und berechnen dann Verzerrung und Varianz. Man beachte, dass dies im realen Leben nicht möglich ist, da wir weder $f(\cdot)$ noch die Parameter des zugefügten Rauschens kennen. Dann lässt sich $E[g(x)]$ mit Hilfe des Durchschnitts von $g_i(\cdot)$ schätzen:

$$\overline{g}(x) = \frac{1}{M}\sum_{i=1}^{M} g_i(x) .$$

Die geschätzte Verzerrung und Varianz ist

$$\begin{aligned} \text{Verzerrung}^2(g) &= \frac{1}{N}\sum_t [\overline{g}(x^t) - f(x^t)]^2 , \\ \text{Varianz}(g) &= \frac{1}{NM}\sum_t \sum_i [g_i(x^t) - \overline{g}(x^t)]^2 . \end{aligned}$$

Betrachten wir nun einige Modelle unterschiedlicher Komplexität. Das einfachste ist eine konstante Funktion:

$$g_i(x) = 2 .$$

Dieses Modell hat keine Varianz, da wir die Daten nicht nutzen und alle $g_i(x)$ gleich sind. Die Verzerrung ist jedoch sehr hoch, es sei denn

natürlich, dass $f(x)$ nahe 2 für alle x liegt. Nehmen wir den Durchschnitt von r^t in der Stichprobe

$$g_i(x) = \sum_t r_i^t / N$$

anstelle der Konstanten 2, so verringert sich dadurch die Verzerrung, da wir erwarten würden, dass der Durchschnitt im Allgemeinen eine bessere Schätzung liefert. Dadurch erhöht sich allerdings die Varianz aufgrund der Tatsache, dass die verschiedenen Stichproben $\mathcal{X}_i$ unterschiedliche Durchschnittswerte haben würden. Normalerweise würde in diesem Fall die Verringerung der Verzerrung stärker sein als die Zunahme an Varianz; der Fehler würde sich ebenfalls verringern.

Im Zusammenhang mit der Polynomregression findet sich ein Beispiel in Abbildung 4.5. Mit zunehmendem Grad des Polynoms verursachen geringe Veränderungen im Datensatz eine stärkere Veränderung in den angepassten Polynomen; somit erhöht sich die Varianz. Ein komplexes Modell erlaubt jedoch im Mittel eine bessere Anpassung an die zugrundeliegende Funktion; daher verringert sich die Verzerrung (siehe Abbildung 4.6). Dies nennt man das *Verzerrung/Varianz-Dilemma*, was für jedes maschinelle Lernsystem zutrifft und nicht nur für die polynomiale Regression (Geman, Bienenstock und Doursat 1992). Um die Verzerrung zu verringern, sollte das Modell flexibel sein, auf Kosten des Risikos einer höheren Varianz. Wird die Varianz niedrig gehalten, ist es uns möglicherweise nicht möglich, eine gute Anpassung an die Daten vorzunehmen und wir erhalten eine starke Verzerrung. Das optimale Modell ist das, welches den besten Kompromiss zwischen Verzerrung und Varianz findet.

VERZERRUNG/ VARIANZ-DILEMMA

Auftretende Verzerrung ist ein Zeichen dafür, dass unsere Modellklasse die Lösung nicht enthält; dies bezeichnet man als *Unteranpassung*. Bei bestehender Varianz ist die Modellklasse zu allgemein gehalten und lernt auch das Rauschen; hierbei spricht man von *Überanpassung*. Wenn $g(\cdot)$ und $f(\cdot)$ aus derselben Hypothesenklasse stammen, beispielsweise einem Polynom gleichen Grades, so haben wir einen verzerrungsfreien Schätzer und die geschätzte Verzerrung verringert sich mit zunehmender Zahl an Modellen. Hierbei wird der fehlerreduzierende Effekt bei korrekter Modellwahl sichtbar (was wir in Kapitel 2 mit *induktiver Verzerrung* bezeichneten – die zwei Verzerrungsarten sind unterschiedlich aber verwandt). Auch die Varianz hängt von der Größe des Datensatzes ab; die Variabilität aufgrund der Stichprobe verringert sich mit zunehmender Stichprobengröße. Zusammenfassend lässt sich sagen, dass ein niedriger Fehlerwert erzielt wird, wenn die entsprechende induktive Verzerrung vorliegt (um eine niedrige Verzerrung im statistischen Sinne zu gewährleisten) und wir einen ausreichend großen Datensatz verwenden, so dass die Variabilität des Modells mit Hilfe der Daten im Zaum gehalten werden kann.

UNTERANPASSUNG

ÜBERANPASSUNG

Man beachte, dass bei großer Varianz die Verzerrung niedrig ist. Dies deutet darauf hin, dass $\overline{g}(x)$ ein guter Schätzer ist. Um einen kleinen

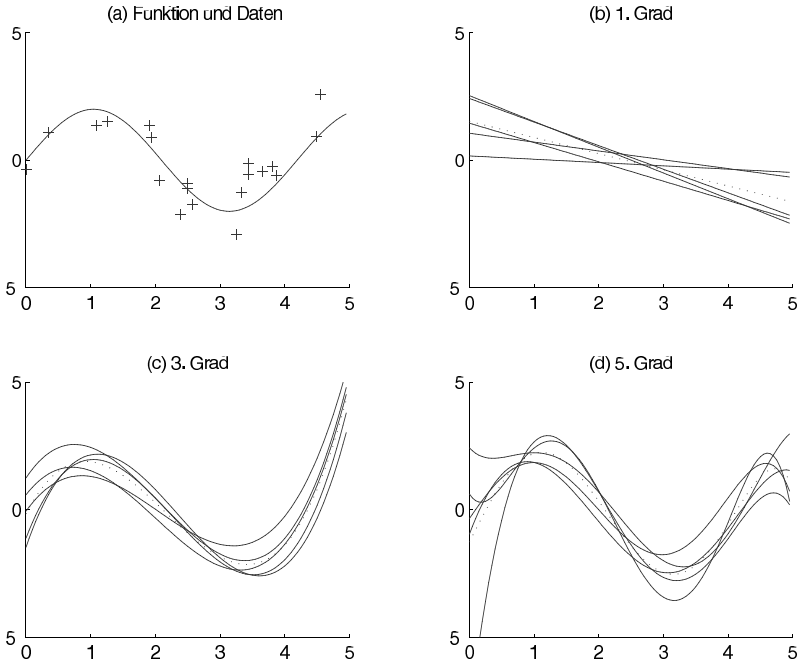


Abb. 4.5: (a) Funktion $f(x) = 2\sin(1{,}5x)$ und ein verrauschter $(\mathcal{N}(0{,}1))$-Datensatz, der nach der Funktion gezogen wurde. Fünf Stichproben werden gezogen; jede enthält 20 Instanzen. (b), (c), (d) sind fünf polynomiale Anpassungen, nämlich $g_i(\cdot)$ des 1., 3. und 5. Grades. Für jeden Fall stellt die gepunktete Linie den Durchschnitt, nämlich $\overline{g}(\cdot)$, der fünf Anpassungen dar.

Fehlerwert zu erhalten, können wir eine große Anzahl an Modellen mit hoher Varianz nutzen und deren Durchschnitt als unseren Schätzer einsetzen. Derlei Ansätze für die Kombination von Modellen werden wir in Kapitel 17 besprechen.

4.8 Modellauswahl

Es gibt eine Zahl an Prozeduren, die wir für die Feinabstimmung der Modellkomplexität nutzen können.

KREUZVALIDIERUNG

Die Methode, auf die wir in der Praxis zurückgreifen, um die optimale Komplexität zu bestimmen, ist das *Kreuzvalidierungsverfahren*. Wir können zwar nicht die Verzerrung und Varianz für ein Modell berechnen, den gesamten Fehler aber schon. Einen vorliegenden Datensatz zerlegen wir in zwei Teile als Trainings- und als Validierungsdatensatz, üben in Frage kommende Modelle unterschiedlicher Komplexität und testen deren Fehler auf dem Validierungssatz, der beim Üben außen vor gelassen wird.

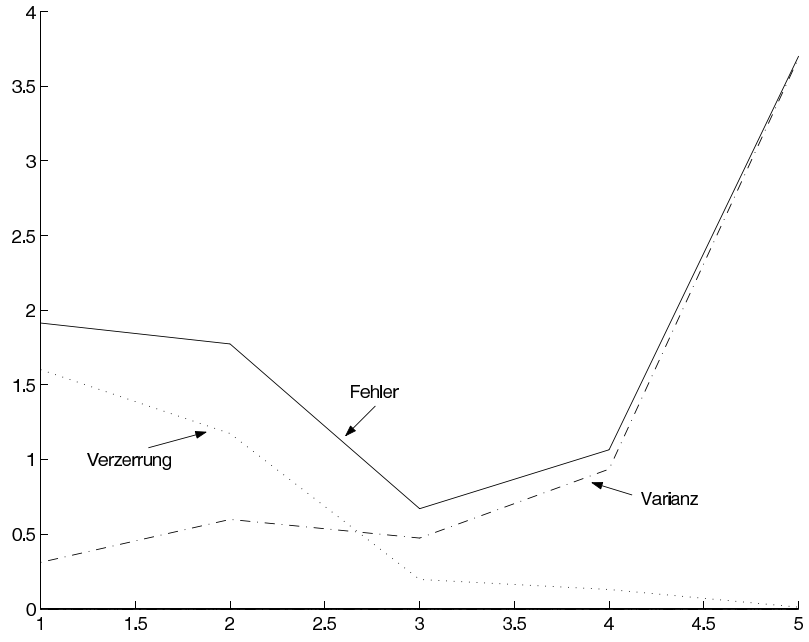


***Abb. 4.6*:** Abgebildet sind unter den gleichen Voraussetzungen wie in Abbildung 4.5 bei Verwendung von 100 statt 5 Modellen die Verzerrung, Varianz und der Fehler für Polynome des 1. bis 5. Grades. Der 1. Grad besitzt die niedrigste Varianz. Der 5. Grad weist die geringste Verzerrung auf. Wenn der Grad erhöht wird, verringert sich die Verzerrung, die Varianz jedoch nimmt zu. Der 3. Grad besitzt den minimalen Fehler.

Mit zunehmender Modellkomplexität verringert sich der Fehler auf dem Trainingsdatensatz. Der Fehler auf dem Validierungsdatensatz nimmt bis zu einem gewissen Grad der Komplexität ab und hört dann entweder auf sich zu verringern oder verringert sich zumindest nicht mehr signifikant; wenn die Daten verrauscht sind, kann er sogar wieder zunehmen. Dieser „Ellenbogen“ entspricht dem optimalen Grad der Komplexität (siehe Abbildung 4.7).

In der Praxis können wir die Verzerrung und somit auch den Fehler nicht so berechnen wie in Abbildung 4.6. Der Validierungsfehler in Abbildung 4.7 ist eine Schätzung hierfür mit dem Unterschied, dass er auch die Varianz des Rauschens beinhaltet: Selbst wenn wir das richtige Modell haben, in dem es keine Verzerrung gibt, und wenn der Datenumfang so groß ist, dass die Varianz vernachlässigbar ist, kann es dennoch einen von null verschiedenen Validierungsfehler geben. Man beachte, dass der Validierungsfehler in Abbildung 4.7 nicht V-förmig ist wie der Fehler in Abbildung 4.6, da dieser mehr Trainingsdaten verwendet, denn wie wir wissen, kann die Varianz durch mehr Daten beschränkt werden. Tatsäch-

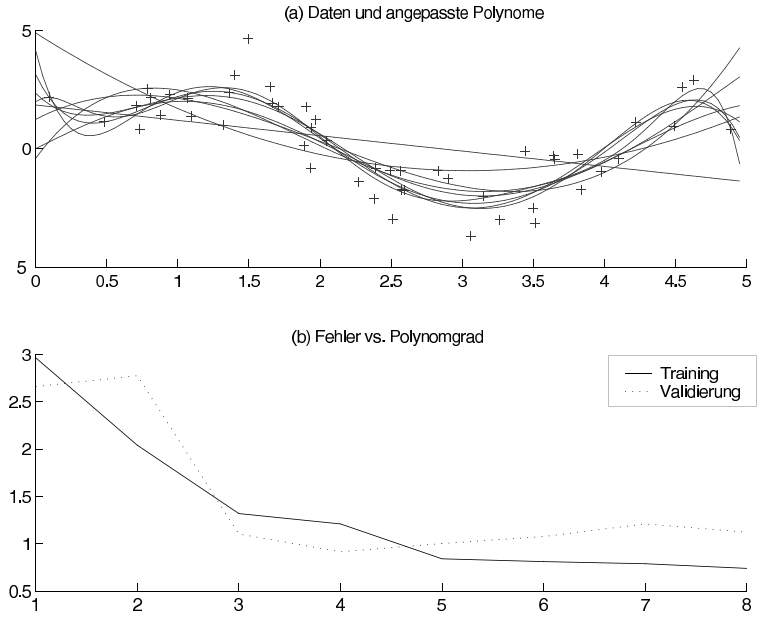


***Abb. 4.7*:** Unter den gleichen Voraussetzungen wie in Abbildung 4.5 werden Trainings- und Validierungsdatensatz (beide mit jeweils 50 Instanzen) generiert. (a) Trainingsdaten und angepasste Polynome 1. bis 8. Grades. (b) Trainings- und Validierungsfehler als Funktion des Polynomgrades. Der „Ellenbogen" liegt bei 3.

lich sehen wir in Abbildung 4.5d, dass sich auch das Polynom fünften Grades dort, wo es Daten gibt, wie ein Polynom dritten Grades verhält. Man beachte, dass es in der Umgebung der beiden Extrema, wo es weniger Datenpunkte gibt, nicht so genau ist.

REGULARISIERUNG

Ein weiterer Ansatz, der häufig genutzt wird, ist die *Regularisierung* (Breiman 1998). Bei diesem Ansatz schreiben wir die *erweiterte Fehlerfunktion*

$$E' = \text{Fehler auf den Daten} + \lambda \cdot \text{Modelkomplexität} \,. \tag{4.37}$$

Hier liegt ein zweiter Term vor, der komplexe Modelle mit großer Varianz straft, wobei λ die Stärke der Bestrafung liefert. Wenn wir die erweiterte Fehlerfunktion statt nur des Fehlers auf den Daten minimieren, strafen wir komplexe Modelle und verringern somit die Varianz. Wenn λ zu groß gewählt wird, so erlaubt dies nur die Nutzung sehr einfacher Modelle und wir riskieren, dass sich Verzerrungen einschleichen. λ wird durch die Kreuzvalidierung optimiert.

Eine andere Betrachtungsweise von Gleichung 4.37 ergibt sich, wenn wir E' als den Fehler innerhalb neuer Testdaten ansehen. Der erste Term

auf der rechten Seite ist der Trainingsfehler und der zweite Term ist ein Optimismusterm, der die Abweichung zwischen dem Trainings- und dem Testfehler schätzt (Hastie, Tibshirani und Friedman 2011). Verfahren wie *Akaikes Informationskriterium* (AIC) und das *Bayessche Informationskriterium* (BIC) basieren darauf, diesen Optimismusterm zu schätzen, ihn zum Trainingsfehler zu addieren und so den Testfehler zu schätzen, wobei es keine Notwendigkeit zur Validierung gibt. Die Größe des Optimismusterms wächst linear mit d, der Anzahl der Eingaben (hier ist sie $k+1$), und sie fällt, wenn die Größe N der Trainingsmenge wächst. Außerdem wächst sie mit σ^2, der Varianz des hinzugefügten Rauschens (diese können wir anhand des Fehlers eines Modells mit geringer Verzerrung schätzen). Für Modelle, die nicht linear sind, sollte d durch die „effektive" Anzahl der Parameter ersetzt werden.

AIC
BIC

Bei der *strukturellen Risikominimierung* (SRM) (Vapnik 1995) nutzt man eine Menge an Modellen, die hinsichtlich ihrer Komplexität geordnet sind. Zum Beispiel Polynome zunehmenden Grades. Die Komplexität ist im Allgemeinen durch die Anzahl an freien Parametern gegeben. Die VC-Dimension ist eine weitere Kennzahl für die Modellkomplexität. In Gleichung 4.37 können wir eine Menge an abnehmenden λ_i nutzen, um eine Menge an nach zunehmender Komplexität geordneten Modellen zu erhalten. Die Modellauswahl durch die SRM entspricht demnach dem Finden desjenigen Modells, das hinsichtlich seines Komplexitätsgrades am einfachsten und bezüglich des empirischen Fehlers an den Daten am besten ist.

STRUKTURELLE RISIKOMINIMIERUNG

Die *minimale Beschreibungslänge* (engl. *minimal description length*, MDL) (Rissanen 1978, Grünwald 2007) nutzt eine informationstheoretische Kenngröße. Die *Kolmogorow-Komplexität* eines Datensatzes definiert sich als die kürzeste Beschreibung der Daten. Sind die Daten einfach, haben sie eine kurze Beschreibung; wenn es sich zum Beispiel um eine Sequenz an Nullen handelt, so können wir das Ganze einfach als Null plus die Länge der Sequenz notieren. Wenn die Daten völlig zufällig sind, dann werden wir keine Beschreibung der Daten finden, die kürzer ist als die Daten selbst. Ist ein Modell den Daten angemessen, dann weist es eine gute Anpassung an die Daten auf, und statt der Daten können wir die Modellbeschreibung senden/speichern. Aus allen Modellen, welche die Daten beschreiben, suchen wir das einfachste Modell, so dass wir damit auch die kürzeste Beschreibung erhalten. Damit schließen wir erneut einen Kompromiss zwischen der Einfachheit eines Modells und seiner Fähigkeit, die Daten gut zu erklären.

MINIMALE BESCHREIBUNGSLÄNGE

Die *Bayessche Modellauswahl* wird genutzt, wenn uns a priori Informationen über die geeignete Klasse an Approximationsfunktionen vorliegen. Dieses Vorabwissen definiert sich als eine a-priori-Verteilung über Modelle, $p(\text{Modell})$. Für vorliegende Daten und unter der Annahme eines Modells können wir $p(\text{Modell}|\text{Daten})$ berechnen, indem wir den Satz von Bayes

BAYESSCHE MODELLAUSWAHL

benutzen:

$$p(\text{Modell}|\text{Daten}) = \frac{p(\text{Daten}|\text{Modell})p(\text{Modell})}{p(\text{Daten})} . \tag{4.38}$$

$p(\text{Modell}|\text{Daten})$ ist die a-posteriori-Wahrscheinlichkeit des Modells mit unserem subjektiven Vorabwissen über die Modelle, sprich $p(\text{Modell})$, und dem objektiven Support, der durch die Daten gewährt wird, also $p(\text{Daten}|\text{Modell})$. Wir können dann das Modell mit der höchsten a-posteriori-Wahrscheinlichkeit wählen oder den Durchschnitt aller Modelle, gewichtet durch ihre a-posteriori-Wahrscheinlichkeiten, berechnen. Wir werden den Bayesschen Ansatz in Kapitel 16 ausführlich besprechen.

Wenn wir in Gleichung 4.38 den Logarithmus bilden, erhalten wir

$$\log p(\text{Modell}|\text{Daten}) = \log p(\text{Daten}|\text{Modell}) + \log p(\text{Modell}) - c . \tag{4.39}$$

Dies hat dieselbe Form wie Gleichung 4.37; die Log-Likelihood der Daten ist der Trainingsfehler und der Logarithmus der a-priori-Wahrscheinlichkeit ist der Strafterm. Wenn wir beispielsweise ein Regressionsmodell haben und die a-priori-Wahrscheinlichkeit $p(\boldsymbol{w}) \sim \mathcal{N}(0, 1/\lambda)$ verwenden, dann minimieren wir

$$E = \sum_t \left[r^t - g(x^t|\boldsymbol{w})\right]^2 + \lambda \sum_i w_i^2 . \tag{4.40}$$

Das heißt, wir suchen nach den w_i, die den Fehler verringern und dabei gleichzeitig so nahe wie möglich bei 0 liegen. Letzteres fordern wir deshalb, weil das angepasste Polynom dann glatter ist. Wenn der Grad des Polynoms zunimmt, um eine bessere Anpassung an die Daten zu erreichen, dann wird die Funktion auf und ab gehen, was bedeutet dass sich die Koeffizienten von 0 entfernen (siehe Abbildung 4.8). Daher wird durch das Hinzufügen dieses Strafterms eine flachere, glattere Anpassung erzwungen. Wie stark die Bestrafung ist, hängt von λ ab, dem Inversen der Varianz der a-priori-Wahrscheinlichkeit – also davon, wie weit weg von 0 wir die Gewichte a priori erwarten. Mit anderen Worten, eine solche a-priori-Wahrscheinlichkeit zu haben, ist äquivalent damit, Parameterwerte nahe 0 zu erzwingen. Wir werden dies in Kapitel 16 ausführlicher diskutieren.

Wenn daher die a-priori-Wahrscheinlichkeit so gewählt wurde, dass wir höhere Wahrscheinlichkeiten den einfacheren Modellen zuweisen (gemäß Ockhams Rasiermesser), sind der Bayessche Ansatz, die Regularisierung, die SRM und die MDL äquivalent. Die Kreuzvalidierung unterscheidet sich dadurch von allen anderen Methoden der Modellauswahl, dass sie keine a-priori-Vermutungen über das Modell anstellt. Wenn genügend Validierungsdaten existieren, ist dies der beste Ansatz. Die anderen Methoden erweisen sich als nützlich, wenn die Datenstichprobe gering ist.

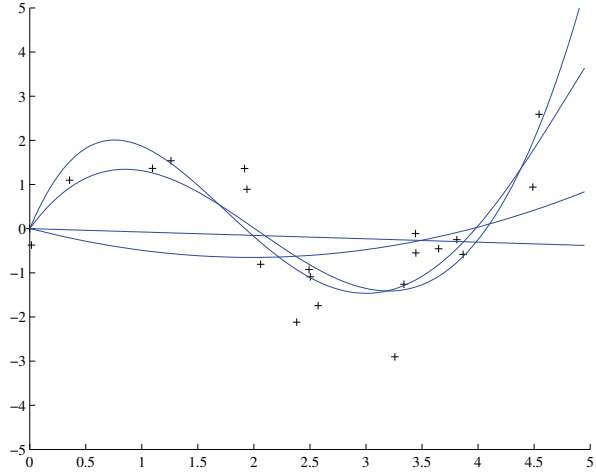


***Abb. 4.8*:** Unter den gleichen Bedingungen wie in Abbildung 4.5 werden Polynome vom Grad 1 bis 4 angepasst. Die Beträge der Koeffizienten wachsen mit dem Grad des Polynoms. Die Werte sind für Grad 1 $[-0{,}0769, 0{,}0016]^T$, für 2 $[0{,}1682, -0{,}6657, 0{,}0080]^T$, für 3 $[0{,}4238, -2{,}5778, 3{,}4675, -0{,}0002]^T$ und für 4 $[-0{,}1093, 1{,}4356, -5{,}5007, 6{,}0454, -0{,}0019]^T$.

4.9 Anmerkungen

Eine gute Quelle zu Maximum-Likelihood-Schätzungen und Bayesschen Schätzungen ist Ross 1987. Viele Lehrbücher zum Thema Mustererkennung behandeln die Klassifikation mit parametrischen Modellen (z.B MacLachlan 1992; Devroye, Györfi und Lugosi 1996; Webb und Copsey 2011; Duda, Hart und Stork 2001). Tests zur Überprüfung auf Vorhandensein univariater Normalität finden sich bei Rencher 1995.

Geman, Bienenstock und Doursat (1992) besprechen die Auflösung von Verzerrung und Varianz für verschiedene Lernmodelle, denen wir uns in nachfolgenden Kapiteln widmen werden. Die Auflösung von Verzerrung und Varianz gilt für die Summe der Verlustquadrate und betrifft die Regression. Eine so komfortable additive Aufspaltung des Fehlers in Verzerrung, Varianz und Rauschen ist für den 0/1-Verlust nicht möglich, da bei der Klassifikation nur dann ein Fehler vorliegt, wenn wir uns zufällig auf die andere Seite der Grenze bewegen. Wenn bei einem Zweiklassenproblem die korrekte a-posteriori-Wahrscheinlichkeit 0,7 ist und unsere Schätzung 0,8, dann liegt kein Fehler vor; einen Fehler haben wir nur dann, wenn unsere Schätzung kleiner als 0,5 ist. Verschiedene Forscher haben Definitionen der Verzerrung und Varianz für die Klassifikation vorgeschlagen; in Friedman 1997 ist eine Übersicht zu finden.

4.10 Übungen

1. Schreiben Sie den Code, der eine Bernoulli-Stichprobe mit gegebenem Parameter p generiert sowie den Code, der $\hat{p}$ anhand der Stichprobe berechnet.

2. Schreiben Sie die Log-Likelihood für eine multinomiale Stichprobe und beweisen Sie Gleichung 4.6.

3. Schreiben Sie den Code, der eine normalverteilte Stichprobe mit gegebenem μ und σ generiert sowie den Code, der m und s anhand der Stichprobe berechnet. Führen Sie selbiges unter Nutzung des Bayesschen Schätzers durch, und zwar unter der Annahme einer a-priori-Verteilung für μ.

4. Gegeben seien zwei Normalverteilungen $p(x|\mathcal{C}_1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ und $p(x|\mathcal{C}_2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$ sowie $P(\mathcal{C}_1)$ und $P(\mathcal{C}_2)$. Berechnen Sie die Bayesschen Diskriminantenpunkte analytisch.

   Lösung: Gegeben ist

   $$p(x|\mathcal{C}_1) \sim \mathcal{N}(\mu_1, \sigma_1^2) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right]$$
   $$p(x|\mathcal{C}_2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

   und wir wollen das x finden, das $P(\mathcal{C}_1|x) = P(\mathcal{C}_2|x)$ erfüllt, also

   $$p(x|\mathcal{C}_1)P(\mathcal{C}_1) = p(x|\mathcal{C}_2)P(\mathcal{C}_2)$$
   $$\log p(x|\mathcal{C}_1) + \log P(\mathcal{C}_1) = \log p(x|\mathcal{C}_2) + \log P(\mathcal{C}_2)$$
   $$-\frac{1}{2}\log 2\pi - \log\sigma_1 - \frac{(x-\mu_1)^2}{2\sigma_1^2} + \log P(\mathcal{C}_1) = \ldots$$
   $$-\log\sigma_1 - \frac{1}{2\sigma_1^2}\left(x^2 - 2x\mu_1 + \mu_1^2\right) + \log P(\mathcal{C}_1) = \ldots$$
   $$\left(\frac{1}{2\sigma_2^2} - \frac{1}{2\sigma_1^2}\right)x^2 + \left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right)x$$
   $$+\left(\frac{\mu_2^2}{2\sigma_2^2} - \frac{\mu_1^2}{2\sigma_1^2}\right) + \log\frac{\sigma_2}{\sigma_1} + \log\frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} = 0\,.$$

   Die letzte Gleichung hat die Form $ax^2 + bx + c = 0$ und ihre beiden Wurzeln sind

   $$x_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}\,.$$

   Wenn die Varianzen gleich sind, verschwindet der quadratische Term und es gibt nur eine Wurzel, d. h., die beiden a-posteriori-Wahrscheinlichkeiten schneiden sich bei einem einzelnen x-Wert.

5. Wie ist das Wahrscheinlichkeitsverhältnis $\frac{p(x|\mathcal{C}_1)}{p(x|\mathcal{C}_2)}$ im Falle von Gaußschen Dichten?

   LÖSUNG:

$$\frac{p(x|\mathcal{C}_1)}{p(x|\mathcal{C}_2)} = \frac{\frac{1}{\sqrt{2\pi}\sigma_1}\exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right]}{\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left[-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right]}.$$

   Im Falle $\sigma_1^2 = \sigma_2^2 = \sigma^2$ vereinfacht sich dies zu

$$\begin{aligned}\frac{p(x|\mathcal{C}_1)}{p(x|\mathcal{C}_2)} &= \exp\left[-\frac{(x-\mu_1)^2}{2\sigma^2} + \frac{(x-\mu_2)^2}{2\sigma^2}\right] \\ &= \exp\left[\frac{(\mu_1-\mu_2)}{\sigma^2} + \frac{(\mu_2^2-\mu_1^2)}{2\sigma^2}\right] \\ &= \exp(wx + w_0)\end{aligned}$$

   mit $w = (\mu_1 - \mu_2)/\sigma^2$ und $w_0 = (\mu_2^2 - \mu_1^2)/2\sigma^2$.

6. Generieren Sie für ein Zweiklassenproblem normalverteilte Stichproben für zwei Klassen mit unterschiedlichen Varianzen. Nutzen Sie dann die parametrische Klassifikation, um die Diskriminantenpunkte zu schätzen. Vergleichen Sie diese mit den theoretischen Werten.

7. Gehen Sie von einem linearen Modell aus und fügen Sie Gaußsches Rauschen mit Nullmittelwert hinzu, um eine Stichprobe zu generieren. Zerlegen Sie Ihre Stichprobe in zwei Teile, einen als Trainingsdatensatz und einen als Validierungsdatensatz. Nutzen Sie die lineare Regression mit dem Trainingsdatensatz. Berechnen Sie den Fehler am Validierungsdatensatz. Wiederholen Sie dies für Polynome 2. und 3. Grades.

8. Wenn der Trainingsdatensatz zu klein ist, könnte der Beitrag der Varianz zum Fehler höher sein als der von der Verzerrung; in solch einem Fall sollten wir ein einfaches Modell möglicherweise vorziehen, auch wenn wir wissen, dass es für diese Aufgabe zu einfach ist. Können Sie ein Beispiel geben?

9. Nehmen wir an, dass wir bei gegebenen Stichproben $\mathcal{X}_i = \{x_i^t, r_i^t\}$ nun $g_i(x) = r_i^1$ definieren, also, dass unsere Schätzung für ein beliebiges x der r-Wert der ersten Instanz im (ungeordneten) Datensatz $\mathcal{X}_i$ ist. Welche Aussagen können Sie hier zur Verzerrung und Varianz treffen, verglichen mit $g_i(x) = 2$ und $g_i(x) = \sum_t r_i^t/N$? Was, wenn eine geordnete Stichprobe vorliegt, so dass $g_i(x) = \min_t r_i^t$?

10. Wie wirkt es sich auf die Verzerrung und die Varianz aus, wenn wir in Gleichung 4.40 den Parameter λ ändern?

Lösung: Der Parameter λ kontrolliert die Glätte. Wenn er groß ist, kann es sein, dass wir zu stark glätten und die Varianz auf Kosten zunehmender Verzerrung verringern. Ist λ klein, haben wir möglicherweise eine geringe Verzerrung, jedoch eine große Varianz.

4.11 Literaturangaben

Andrieu, C., N. de Freitas, A. Doucet, and M. I. Jordan. 2003. „An Introduction to MCMC for Machine Learning." *Machine Learning* 50: 5–43.

Breiman, L. 1998. „Bias-Variance, Regularization, Instability and Stabilization." In *Neural Networks and Machine Learning,* ed. C. M. Bishop, 27–56. Berlin: Springer.

Devroye, L., L. Györfi, and G. Lugosi. 1996. *A Probabilistic Theory of Pattern Recognition.* New York: Springer.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Friedman, J. H. 1997. „On Bias, Variance, 0/1-Loss and the Curse of Dimensionality." *Data Mining and Knowledge Discovery* 1: 55–77.

Geman, S., E. Bienenstock, and R. Doursat. 1992. „Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4: 1–58.

Grünwald, P. D. 2007. *The Minimum Description Length Principle.* Cambridge, MA: MIT Press.

Hastie, T., R. Tibshirani, and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* 2nd ed. New York: Springer.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Rissanen, J. 1978. „Modeling by Shortest Data Description." *Automatica* 14: 465–471.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Webb, A., and K. D. Copsey 2011. *Statistical Pattern Recognition.* 3rd ed. New York: Wiley.

5 Multivariate Methoden

In Kapitel 4 haben wir den parametrischen Ansatz bei der Klassifikation und Regression diskutiert. Nun verallgemeinern wir dies auf den multivariaten Fall, bei dem uns multiple Eingaben vorliegen, und bei dem die Ausgabe, die ein Klassencode oder eine kontinuierliche Ausgabe ist, eine Funktion dieser multiplen Eingaben darstellt. Diese Eingabewerte können diskrete oder numerische Werte sein. Wir werden uns damit befassen, wie solche Funktionen anhand einer multivariaten Stichprobe, für deren Elemente die Ausgabewerte bekannt sind, erlernt werden können. Außerdem diskutieren wir, wie die Komplexität des Lerners an die vorliegenden Daten feinangepasst werden kann.

5.1 Multivariate Daten

Bei vielen Anwendungen werden diverse Messungen für jedes einzelne Individuum bzw. Ereignis durchgeführt. Die Ergebnisse werden in Form von *Beobachtungsvektoren* zusammengefasst:

$$\mathbf{X} = \begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_d^1 \\ X_1^2 & X_2^2 & \cdots & X_d^2 \\ \vdots & & & \\ X_1^N & X_2^N & \cdots & X_d^N \end{bmatrix},$$

wobei die d Spalten den d *Variablen* entsprechen, welche die Ergebnisse der Messungen zu einem Individuum oder Ereignis darstellen. Diese nennt man auch *Eingaben*, *Merkmale* oder *Attribute*. Die N Zeilen entsprechen unabhängigen und identisch verteilten *Beobachtungen*, *Beispielen* oder *Instanzen* zu N Individuen bzw. Ereignissen.

EINGABE

MERKMAL

ATTRIBUT

Bei der Bearbeitung von Darlehensanträgen beispielsweise besteht ein Beobachtungsvektor aus den Informationen zu einem bestimmten Kunden. Er enthält u.a. das Alter, den Familienstand, das jährliche Einkommen, und so weiter, und wir betrachten N solche Kunden. Diese Messwerte können durchaus unterschiedliche Skalen haben; so wird zum Beispiel das Alter in Jahren und das jährliche Einkommen in einer Währungseinheit

BEOBACHTUNG
BEISPIEL

INSTANZ

erhoben. Einige Werte, wie das Alter, können numerisch sein, wohingegen andere, wie der Familienstand, vom diskreten Typ sein könnten.

Typischerweise sind diese Variablen korreliert. Falls sie es nicht sind, so gibt es keinen Grund für eine multivariate Analyse. Unser Ziel kann in der *Vereinfachung* bestehen, das heißt in der Zusammenfassung dieser großen Datenmenge mit Hilfe von relativ wenigen Parametern. Unser Ziel kann auch *exploratorisch* sein; möglicherweise interessiert uns die Generierung von Hypothesen zu den Daten. Bei einigen Anwendungen interessiert uns die Vorhersage des Wertes einer Variable anhand der Werte von anderen Variablen. Wenn die vorhergesagte Variable diskret ist, so spricht man von einer multivariaten Klassifikation, und bei einem numerischen Wert handelt es sich um ein multivariates Regressionsproblem.

5.2 Parameterschätzung

Mittelvektor

Der *Mittelwertvektor* $\boldsymbol{\mu}$ definiert sich daraus, dass jedes seiner Elemente der Mittelwert einer Spalte von $\mathbf{X}$ ist:

$$E[\boldsymbol{x}] = \boldsymbol{\mu} = [\mu_1, \ldots, \mu_d]^T \,. \tag{5.1}$$

Die Varianz von X_i wird mit σ_i^2 bezeichnet. Die Kovarianz zweier Variablen X_i und X_j ist definiert als

$$\sigma_{ij} \equiv \mathrm{Cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] = E[X_i X_j] - \mu_i \mu_j \,, \tag{5.2}$$

wobei $\sigma_{ij} = \sigma_{ji}$ und falls $i = j$, dann gilt $\sigma_{ii} = \sigma_i^2$. Bei d Variablen gibt es d Varianzen und $d(d-1)/2$ Kovarianzen, die im Allgemeinen durch eine Matrix der Form $d \times d$ repräsentiert werden; diese bezeichnet man auch als *Kovarianzmatrix*, dargestellt durch $\boldsymbol{\Sigma}$, deren (i, j)-tes Element σ_{ij} ist:

Kovarianzmatrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & & & \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix} .$$

Die diagonalen Terme sind die Varianzen, die Nichtdiagonalterme entsprechen den Kovarianzen und die Matrix ist symmetrisch. In der Vektormatrixnotation erhalten wir:

$$\boldsymbol{\Sigma} \equiv \mathrm{Cov}(\boldsymbol{X}) = E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T] = E[\boldsymbol{X}\boldsymbol{X}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T \,. \tag{5.3}$$

Wenn zwei Variablen in einer linearen Beziehung stehen, so ergibt sich eine positive oder negative Kovarianz in Abhängigkeit davon, ob die

Beziehung einen positiven oder negativen Anstieg besitzt. Die Größe der Beziehung ist jedoch nur schwierig zu interpretieren, da sie von den Einheiten, in denen die beiden Variablen gemessen werden, abhängt. Die *Korrelation* zwischen zwei Variablen X_i und X_j ist eine Statistik, die zwischen -1 und $+1$ normalisiert und wie folgt definiert ist:

KORRELATION

$$\text{Corr}(X_i, X_j) \equiv \rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} . \tag{5.4}$$

Wenn zwei Variablen unabhängig sind, ist ihre Kovarianz und damit auch ihre Korrelation gleich null. Die umgekehrte Behauptung lässt sich jedoch nicht aufstellen. Variablen können abhängig sein (auf nichtlineare Weise) und ihre Korrelation kann dennoch null sein.

Bei gegebener multivariater Stichprobe können Schätzungen dieser Parameter wie folgt berechnet werden. Der Maximum-Likelihood-Schätzer für den Mittelwert entspricht dem *Stichprobenmittelwert* $\boldsymbol{m}$. Seine i-te Dimension ist der Durchschnitt der i-ten Spalte von $\mathbf{X}$:

STICHPROBEN-MITTELWERT

$$\boldsymbol{m} = \frac{\sum_{t=1}^{N} \boldsymbol{x}^t}{N} \text{ mit } m_i = \frac{\sum_{t=1}^{N} x_i^t}{N}, i = 1, \ldots, d . \tag{5.5}$$

Der Schätzer von $\boldsymbol{\Sigma}$ ist $\mathbf{S}$, die *Stichprobenkovarianzmatrix*, mit den Einträgen

STICHPROBEN-KOVARIANZ

$$s_i^2 = \frac{\sum_{t=1}^{N} (x_i^t - m_i)^2}{N} \tag{5.6}$$

$$s_{ij} = \frac{\sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)}{N} . \tag{5.7}$$

Dies sind verzerrte Schätzungen; wenn aber in einer Anwendung die Schätzungen signifikant variieren, je nachdem, ob wir durch N oder $N-1$ dividieren, dann haben wir nichtsdestotrotz ernsthafte Probleme.

Die *Stichprobenkorrelationskoeffizienten* sind

STICHPROBEN-KORRELATION

$$r_{ij} = \frac{s_{ij}}{s_i s_j} \tag{5.8}$$

und die Stichprobenkorrelationsmatrix $\mathbf{R}$ enthält r_{ij}.

5.3 Schätzung von fehlenden Werten

Häufig fehlen die Werte von bestimmten Variablen bei Beobachtungen. Die beste Strategie ist die, derlei Beobachtungen komplett zu verwerfen;

allerdings haben wir im Allgemeinen keine ausreichend großen Stichproben, um uns dies erlauben zu können und wir wollen Daten auch nicht verlieren, da die vorhandenen Einträge durchaus Informationen enthalten. Wir versuchen, die fehlenden Einträge durch Schätzung derselben zu ergänzen. Dieses Vorgehen nennt man *Imputation*.

Imputation

Bei der *Mittelwertimputation* nutzen wir für eine numerische Variable den Mittelwert (Durchschnitt) der vorhandenen Daten für diese Variable in der Stichprobe als Ersatz. Für eine diskrete Variable fügen wir den wahrscheinlichsten Wert ein, das heißt den Wert, der am häufigsten in den Daten auftaucht.

Bei der *Imputation durch Regression* versuchen wir, den Wert einer fehlenden Variable anhand anderer Variablen vorherzusagen, deren Werte für diesen Fall bekannt sind. In Abhängigkeit vom Typ der fehlenden Variable definieren wir ein separates Modell für dieses Regressions- oder Klassifikationsproblem, das wir anhand der Datenpunkte, für die solche Werte bekannt sind, trainieren. Wenn viele verschiedene Variablen fehlen, nutzen wir die Mittelwerte als anfängliche Schätzungen und führen die Prozedur dann iterativ weiter, bis die vorhergesagten Werte sich stabilisieren. Wenn die Variablen nicht stark korrelieren, ist der Regressionsansatz äquivalent zur Mittelwertimputation.

In Abhängigkeit vom Kontext ist jedoch manchmal die Tatsache, dass ein gewisser Attributwert fehlt, von Bedeutung. Wenn beispielsweise bei einem Kreditkartenantrag ein Kunde seine Telefonnummer nicht angibt, dann ist dies unter Umständen eine äußerst wichtige Information. In solchen Fällen zeigt ein eigens dafür definierter Wert an, dass der eigentliche Wert fehlt; wir arbeiten dann mit diesem definierten Wert weiter.

5.4 Multivariate Normalverteilung

Beim multivariaten Fall, bei dem $\boldsymbol{x}$ d-dimensional und normalverteilt ist, erhalten wir:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{5.9}$$

und wir schreiben $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, wobei $\boldsymbol{\mu}$ der Mittelwertvektor und $\boldsymbol{\Sigma}$ die Kovarianzmatrix ist (siehe Abbildung 5.1). Analog zur Nutzung von

$$\frac{(x-\mu)^2}{\sigma^2} = (x-\mu)(\sigma^2)^{-1}(x-\mu)$$

als quadratische Distanz von x nach μ in Vielfachen der Standardabweichung (was das Normalisieren für verschiedene Varianzen erlaubt), nutzt man im multivariaten Fall den *Mahalanobis-Abstand*:

Mahalanobis-Abstand

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\,. \tag{5.10}$$

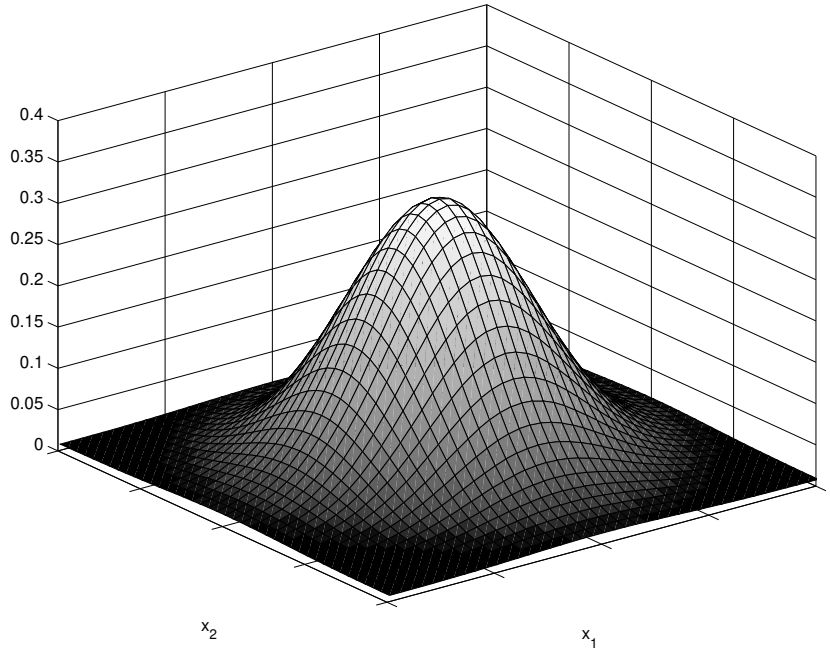


***Abb. 5.1*:** Bivariate Normalverteilung.

$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = c^2$ ist die d-dimensionale Hyperellipse mit Zentrum $\boldsymbol{\mu}$, deren Form und Ausrichtung durch $\boldsymbol{\Sigma}$ definiert wird. Aufgrund der Nutzung der Umkehrung von $\boldsymbol{\Sigma}$ erhält eine Variable weniger Gewicht beim Mahalanobis-Abstand, wenn sie eine größere Varianz als eine andere Variable aufweist. Ähnlich dazu gehen zwei hochkorrelierte Variablen weniger stark in die Berechnung ein, als zwei schwächer korrelierte Variablen. Die Nutzung der Umkehrung der Kovarianzmatrix hat somit den Effekt, alle Variablen auf die Einheitsvarianz zu standardisieren und Korrelationen zu eliminieren.

Betrachten wir nun zum Zwecke der visuellen Verdeutlichung den bivariaten Fall mit $d = 2$ (siehe Abbildung 5.2). Wenn die Variablen unabhängig sind, dann sind die Hauptachsen der Dichte parallel zu den Eingabeachsen. Die Dichte wird zu einer Ellipse, wenn die Varianzen sich unterscheiden. Die Dichte rotiert in Abhängigkeit vom Vorzeichen der Kovarianz (Korrelation). Der Mittelwertvektor ist $\boldsymbol{\mu}^T = [\mu_1, \mu_2]$ und die Kovarianzmatrix wird gewöhnlich ausgedrückt durch:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} .$$

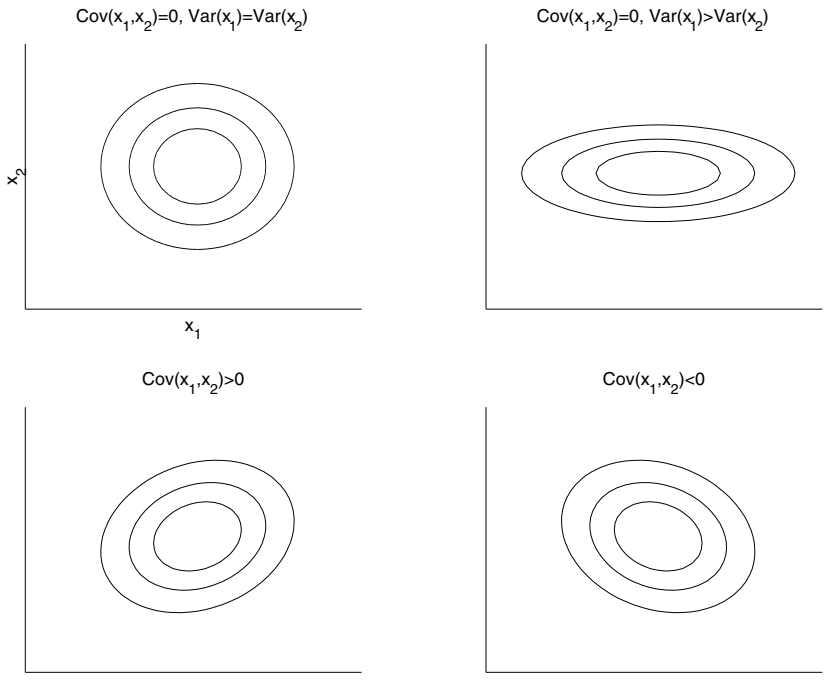


***Abb. 5.2*:** Konturdiagramm vier verschiedener bivariater Normalverteilungen (gezeigt jeweils drei Isowahrscheinlichkeiten). Das Zentrum ergibt sich durch den Mittelwert. Form und Ausrichtung hängen von der Kovarianzmatrix ab.

Die gemeinsame bivariate Dichte kann durch die Form (siehe Übung 1)

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[-\frac{1}{2(1-\rho^2)}\left(z_1^2 - 2\rho z_1 z_2 + z_2^2\right)\right] \tag{5.11}$$

dargestellt werden, wobei $z_i = (x_i - \mu_i)/\sigma_i$, $i = 1, 2$ standardisierten Variablen entsprechen; hierbei spricht man von einer *z-Normalisierung*. Es sei daran erinnert, dass

z-Normalisierung

$$z_1^2 + 2\rho z_1 z_2 + z_2^2 = \text{konstant}$$

für $|\rho| < 1$ die Gleichung einer Ellipse ist. Wenn $\rho > 0$, so hat die Hauptachse der Ellipse einen positiven Anstieg, und wenn $\rho < 0$, hat die Hauptachse einen negativen Anstieg.

Beim erweiterten Mahalanobis-Abstand von Gleichung 5.11 ist jede Variable normalisiert, um Einheitsvarianz zu erzielen, und es gibt einen Kreuzterm, der für Korrekturen der Korrelation zwischen den zwei Variablen zuständig ist.

Die Dichte hängt von fünf Parametern ab: den zwei Mittelwerten, den zwei Varianzen und der Korrelation. $\boldsymbol{\Sigma}$ ist nicht singulär und somit positiv definit, vorausgesetzt, dass die Varianzen ungleich null sind und $|\rho| < 1$. Wenn ρ gleich $+1$ oder -1, stehen die zwei Variablen in linearer Beziehung, die Beobachtungen sind effektiv betrachtet eindimensional, und eine der zwei Variablen kann weggelassen werden. Wenn $\rho = 0$, so sind die zwei Variablen unabhängig, der Kreuzterm verschwindet und wir erhalten ein Produkt zweier univariater Dichten.

Im multivariaten Fall deutet ein geringer Wert von $|\boldsymbol{\Sigma}|$ darauf hin, dass Stichproben nahe $\boldsymbol{\mu}$ sind, genauso wie im univariaten Fall ein geringer Wert für σ^2 darauf hinweist, dass die Stichproben nahe μ liegen. Geringe $|\boldsymbol{\Sigma}|$ können ebenfalls ein Indiz dafür sein, dass eine hohe Korrelation zwischen Variablen besteht. $\boldsymbol{\Sigma}$ ist eine symmetrische, positiv definite Matrix; dies ist der multivariate Ausdruck für $\mathrm{Var}(X) > 0$. Wenn dies nicht der Fall ist, so ist $\boldsymbol{\Sigma}$ singulär und die zugehörige Determinante ist null. Dies liegt entweder an der linearen Abhängigkeit zwischen den Dimensionen oder daran, dass eine der Dimensionen keine Varianz besitzt. In solch einem Fall sollte die Dimensionalität reduziert werden, um eine positiv defnite Matrix zu erhalten; Methoden dafür werden in Kapitel 6 diskutiert.

Wenn $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, so ist jede Dimension von $\boldsymbol{x}$ univariat normalverteilt. (Der Umkehrschluss ist nicht möglich. Jedes X_i kann univariat normalverteilt sein, aber $\boldsymbol{X}$ muss nicht unbedingt multivariat normalverteilt sein.) Um genau zu sein, ist jede Teilmenge mit $k < d$ der Variablen k-variat normalverteilt.

Ein einfacher Spezialfall liegt vor, wenn die Komponenten von $\boldsymbol{x}$ unabhängig sind und $\mathrm{Cov}(X_i, X_j) = 0$ für $i \neq j$ und $\mathrm{Var}(X_i) = \sigma_i^2, \forall i$. Dann ist die Kovarianzmatrix diagonal und die gemeinsame Dichte ist das Produkt der individuellen univariaten Dichten:

$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p_i(x_i) = \frac{1}{(2\pi)^{d/2} \prod_{i=1}^{d} \sigma_i} \exp\left[-\frac{1}{2} \sum_{i=1}^{d} \left(\frac{x_i - \mu_i}{\sigma_i}\right)^2\right] . \quad (5.12)$$

Betrachten wir nun ein weiteres Merkmal, welches wir in nachfolgenden Kapiteln nutzen werden. Sagen wir, dass $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ und $\boldsymbol{w} \in \mathbb{R}^d$; so ergibt sich

$$\boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \sim \mathcal{N}(\boldsymbol{w}^T \boldsymbol{\mu}, \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}) ,$$

vorausgesetzt, dass

$$E[\boldsymbol{w}^T \boldsymbol{x}] = \boldsymbol{w}^T E[\boldsymbol{x}] = \boldsymbol{w}^T \boldsymbol{\mu} , \quad (5.13)$$

$$\begin{aligned} \mathrm{Var}(\boldsymbol{w}^T \boldsymbol{x}) &= E[(\boldsymbol{w}^T \boldsymbol{x} - \boldsymbol{w}^T \boldsymbol{\mu})^2] = E[(\boldsymbol{w}^T \boldsymbol{x} - \boldsymbol{w}^T \boldsymbol{\mu})(\boldsymbol{w}^T \boldsymbol{x} - \boldsymbol{w}^T \boldsymbol{\mu})] \\ &= E[\boldsymbol{w}^T (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{w}] = \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] \boldsymbol{w} \\ &= \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w} . \end{aligned} \quad (5.14)$$

Das heißt, die Projektion einer d-dimensionalen Normalen auf den Vektor $\boldsymbol{w}$ ist univariat normalverteilt. Im allgemeinen Fall, wenn $\mathbf{W}$ eine $d \times k$ Matrix mit Ordnung $k < d$ ist, so ist eine k-dimensionale $\mathbf{W}^T\boldsymbol{x}$ k-variat normalverteilt:

$$\mathbf{W}^T\boldsymbol{x} \sim \mathcal{N}_k(\mathbf{W}^T\boldsymbol{\mu}, \mathbf{W}^T\boldsymbol{\Sigma}\mathbf{W}) \,. \tag{5.15}$$

Die Projektion einer d-dimensionale Normalverteilung in einen k-dimensionalen Raum ergibt also eine k-dimensionale Normalverteilung.

5.5 Multivariate Klassifikation

Für $\boldsymbol{x} \in \mathbb{R}^d$ und wenn die klassenbezogenen Dichten $p(\boldsymbol{x}|\mathcal{C}_i)$ als Normalverteilungen $\mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ angenommen werden, erhalten wir

$$p(\boldsymbol{x}|\mathcal{C}_i) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right] \,. \tag{5.16}$$

Der hauptsächliche Grund hierfür liegt in der analytischen Einfachheit (Duda, Hart und Stork 2001). Außerdem ist die normalverteilte Dichte ein Modell für viele natürlich auftretende Phänomene, da Beispiele der meisten Klassen als leicht veränderte Versionen eines einzelnen Prototyps $\boldsymbol{\mu}_i$ angesehen werden können und die Kovarianzmatrix $\boldsymbol{\Sigma}_i$ die Stärke des Rauschens für jede Variable und die Korrelationen dieser Quellen des Rauschens angibt. Während reale Daten oft nicht exakt multivariat normalverteilt sein müssen, ist es dennoch eine nützliche Approximation. Zusätzlich zu seiner mathematischen Nachvollziehbarkeit ist das Modell robust gegenüber Abweichungen von der Normalität, wie in vielen Arbeiten nachgewiesen wurde (z. B. McLachlan 1992). Allerdings besteht eine klare Voraussetzung darin, dass die Stichprobe einer Klasse eine einzelne Gruppe bilden sollte; wenn es multiple Gruppen gibt, so empfiehlt es sich, auf ein gemischtes Modell zurückzugreifen (Kapitel 7).

Angenommen, wir wollen den Typ eines Autos vorhersagen, für das sich ein Kunde interessieren könnte. Verschiedene Autos stellen die Klassen dar und $\boldsymbol{x}$ steht für die beobachtbaren Daten der Kunden, beispielsweise Alter und Einkommen. $\boldsymbol{\mu}_i$ ist der Vektor des mittleren Alters und Einkommens von Kunden, die ein Auto vom Typ i kaufen und $\boldsymbol{\Sigma}_i$ ist deren Kovarianzmatrix: σ_{i1}^2 und σ_{i2}^2 sind die Varianzen für Alter bzw. Einkommen, und σ_{i12} ist die Kovarianz des Alters und Einkommen in der Gruppe von Kunden, die einen Wagen von Typ i kaufen.

Wenn wir die Diskriminanzfunktion definieren als

$$g_i(\boldsymbol{x}) = \log p(\boldsymbol{x}|\mathcal{C}_i) + \log P(\mathcal{C}_i)$$

und annehmen, dass $p(\boldsymbol{x}|\mathcal{C}_i) \sim \mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, so erhalten wir

$$g_i(\boldsymbol{x}) = -\tfrac{d}{2}\log 2\pi - \tfrac{1}{2}\log|\boldsymbol{\Sigma}_i| - \tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \log P(\mathcal{C}_i) \,. \tag{5.17}$$

Bei gegebener Trainingsstichprobe für $K \geq 2$ Klassen, $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}$, mit $r_i^t = 1$, falls $\boldsymbol{x}^t \in \mathcal{C}_i$ und $r_i^t = 0$ in allen anderen Fällen, findet man Schätzungen für die Mittelwerte und Kovarianzen mit Hilfe der auf jede Klasse separat angewendeten Maximum-Likelihood-Methode:

$$\begin{aligned} \hat{P}(\mathcal{C}_i) &= \frac{\sum_t r_i^t}{N} , \\ \boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} , \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t} . \end{aligned} \tag{5.18}$$

Diese werden dann in die Diskriminanzfunktion eingesetzt, um die Schätzwerte für diese zu erhalten. Unter Nichtbeachtung des ersten konstanten Terms erhalten wir

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i) + \log \hat{P}(\mathcal{C}_i) . \tag{5.19}$$

Erweitern wir dies, so ergibt sich

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}\left(\boldsymbol{x}^T \mathbf{S}_i^{-1}\boldsymbol{x} - 2\boldsymbol{x}^T \mathbf{S}_i^{-1}\boldsymbol{m}_i + \boldsymbol{m}_i^T \mathbf{S}_i^{-1}\boldsymbol{m}_i\right) + \log \hat{P}(\mathcal{C}_i) ,$$

womit eine *quadratische Diskriminanzfunktion* (siehe Abbildung 5.3) definiert wird, die auch in folgender Form geschrieben werden kann:

QUADRATISCHE DISKRIMINANZFUNKTION

$$g_i(\boldsymbol{x}) = \boldsymbol{x}^T \mathbf{W}_i \boldsymbol{x} + \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} , \tag{5.20}$$

mit

$$\begin{aligned} \mathbf{W}_i &= -\frac{1}{2}\mathbf{S}_i^{-1} , \\ \boldsymbol{w}_i &= \mathbf{S}_i^{-1}\boldsymbol{m}_i , \\ w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T \mathbf{S}_i^{-1}\boldsymbol{m}_i - \frac{1}{2}\log|\mathbf{S}_i| + \log \hat{P}(\mathcal{C}_i) . \end{aligned}$$

Die Anzahl an hier zu schätzenden Parameter ist $K \cdot d$ für die Mittelwerte und $K \cdot d(d+1)/2$ für die Kovarianzmatrizen. Bei großem d und kleinen Stichproben kann $\mathbf{S}_i$ singulär und ihre Inverse nicht definiert sein. Möglich ist auch, dass $|\mathbf{S}_i|$ ungleich null aber zu klein ist; in diesem Fall wäre es instabil und geringe Veränderungen in $\mathbf{S}_i$ führen zu starken Veränderungen in $\mathbf{S}_i^{-1}$. Damit die Schätzungen für kleine Stichproben verlässlich sind, ist es ratsam, die Dimensionalität d durch Umgestaltung des Merkmalsextraktors zu verringern und eine Teilmenge der Merkmale auszuwählen oder in irgendeiner Form existierende Merkmale zu kombinieren. Wir werden derlei Methoden in Kapitel 6 diskutieren.

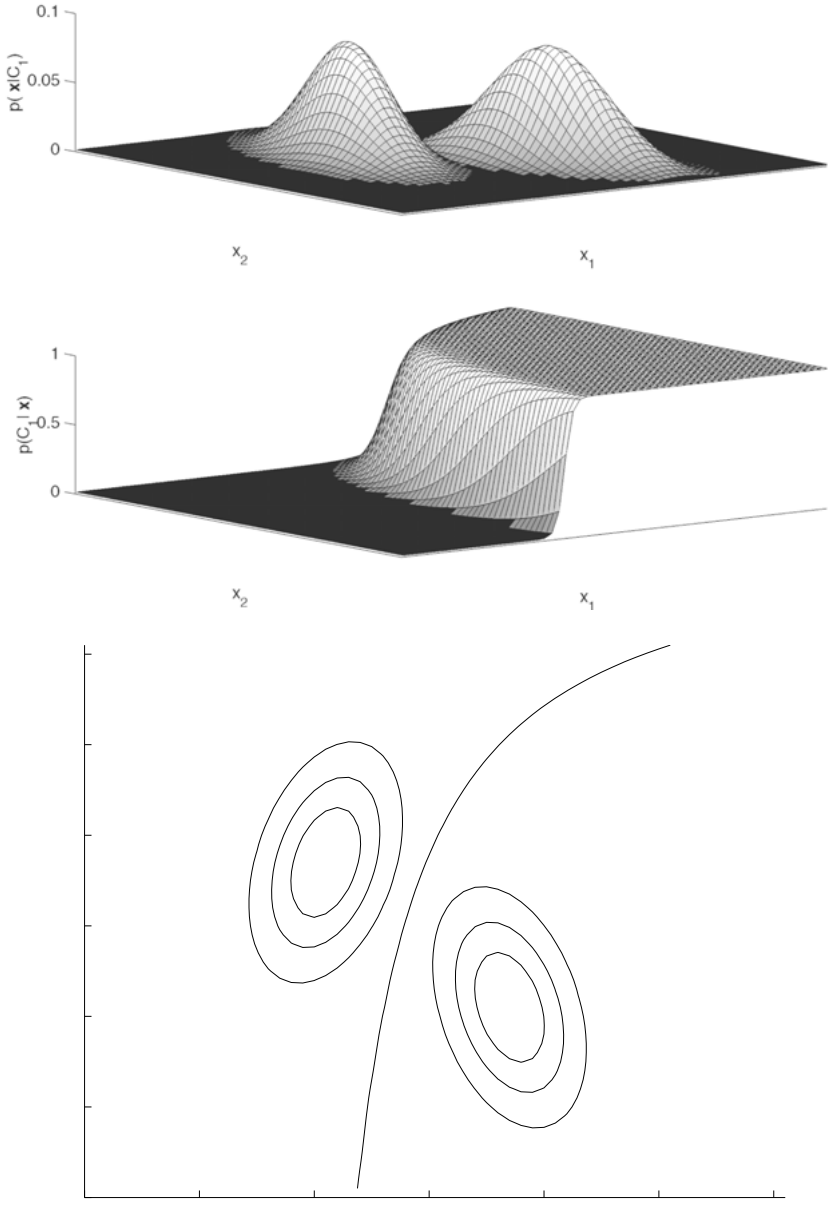


***Abb. 5.3*:** Klassen besitzen verschiedene Kovarianzmatrizen. Abgebildet sind hier die Likelihoodverteilungen und die a-posteriori-Wahrscheinlichkeit für eine der Klassen (oberes Bild). Klassenverteilungen werden durch Isowahrscheinlichkeitskonturdiagramme angedeutet und die Diskriminanzfunktion ist eingezeichnet (unteres Bild).

Eine weitere Möglichkeit besteht darin, die Daten zusammenzuführen und eine gemeinsame Kovarianzmatrix für alle Klassen zu berechnen:

$$\mathbf{S} = \sum_i \hat{P}(\mathcal{C}_i)\mathbf{S}_i \,. \tag{5.21}$$

Wenn die Kovarianzmatrizen gleich sind, dann reduziert sich Gleichung 5.19 auf

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{m}_i)^T\mathbf{S}^{-1}(\boldsymbol{x} - \boldsymbol{m}_i) + \log \hat{P}(\mathcal{C}_i) \,. \tag{5.22}$$

Die Anzahl an Parametern ist $K \cdot d$ für die Mittelwerte und $d(d+1)/2$ für die gemeinsame Kovarianzmatrix. Wenn die a-priori-Wahrscheinlichkeiten der Klassen gleich sind, so lautet die optimale Entscheidungsregel, eine Eingabe der Klasse zuzuteilen, deren Mittelwert den kleinsten Mahalanobis-Abstand zur Eingabe aufweist. Wie schon zuvor, verschieben ungleiche a-priori-Werte die Grenzen in Richtung der weniger wahrscheinlichen Klasse. Man beachte, dass sich in diesem Fall der quadratische Term $\boldsymbol{x}^T\mathbf{S}^{-1}\boldsymbol{x}$ auflöst, da er allen Diskriminanzfunktionen gemeinsam ist, und dass die Entscheidungsgrenzen linear sind und zu einer *linearen Diskriminanzfunktion* (Abbildung 5.4) führen, die wie folgt geschrieben werden kann:

LINEARE DISKRIMINANZFUNKTION

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{5.23}$$

mit

$$\begin{aligned} \boldsymbol{w}_i &= \mathbf{S}^{-1}\boldsymbol{m}_i \,, \\ w_{i0} &= -\frac{1}{2}\boldsymbol{m}_i^T\mathbf{S}^{-1}\boldsymbol{m}_i + \log \hat{P}(\mathcal{C}_i) \,. \end{aligned}$$

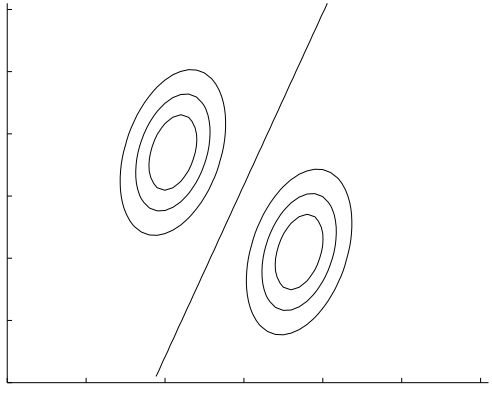


Abb. 5.4: Kovarianzen können willkürlich sein, sind aber beiden Klassen gemein.

Entscheidungsregionen solcher linearer Klassifikatoren sind konvex; sprich, wenn zwei Punkte willkürlich in einer Entscheidungsregion gewählt und durch eine Gerade miteinander verbunden werden, so liegen alle Punkte auf der Geraden in dieser Region.

Weitere Vereinfachung ist unter Umständen dadurch möglich, dass alle Nichtdiagonalelemente der Kovarianzmatrix als null angenommen werden und somit unabhängige Variablen vorausgesetzt werden. Dies ist der *naive Bayessche Klassifikator*, bei dem $p(x_j|C_i)$ univariate Gaußverteilungen sind. $\mathbf{S}$ und ihre Inverse sind diagonal und wir erhalten

NAIVER BAYESSCHER KLASSIFIKATOR

$$g_i(\boldsymbol{x}) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log \hat{P}(\mathcal{C}_i)\,. \tag{5.24}$$

Der Term $((x_j^t - m_{i,j})/s_j)^2$ führt zu einer Normalisierung und misst den Abstand in Vielfachen der Standardabweichung. Geometrisch interpretiert sind die Klassen hyperelliptisch und aufgrund der auf null gesetzten Kovarianzen axial ausgerichtet (siehe Abbildung 5.5). Die Anzahl an Parametern ist $K \cdot d$ für die Mittelwerte und d für die Varianzen. Somit reduziert sich die Komplexität für $\mathbf{S}$ von $\mathcal{O}(d^2)$ auf $\mathcal{O}(d)$.

Noch eine weitere Vereinfachung ergibt sich sogar, wenn wir annehmen, dass alle Varianzen gleich sind; dann reduziert sich der Mahalanobis-Abstand auf die *Euklidische Distanz*. Geometrisch interpretiert ist die Verteilung sphärisch geformt und um den Mittelwertvektor $\boldsymbol{m}_i$ zentriert (siehe Abbildung 5.7). Somit ist $|\mathbf{S}| = s^{2d}$ und $\mathbf{S}^{-1} = (1/s^2)\mathbf{I}$. Die Anzahl an Parametern in diesem Fall ist $K \cdot d$ für die Mittelwerte und einer für s^2.

EUKLIDISCHE DISTANZ

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\frac{\|\boldsymbol{x} - \boldsymbol{m}_i\|^2}{2s^2} + \log \hat{P}(\mathcal{C}_i) \\ &= -\frac{1}{2s^2}\sum_{j=1}^{d}(x_j^t - m_{ij})^2 + \log \hat{P}(\mathcal{C}_i) \end{aligned} \tag{5.25}$$

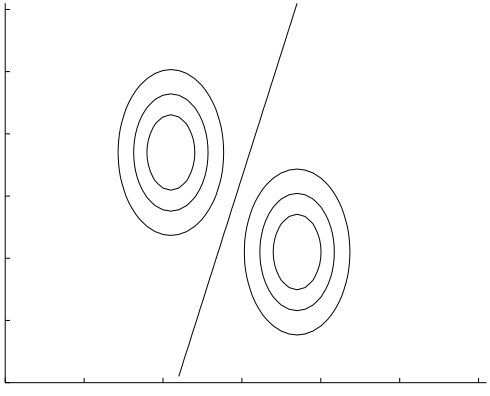


Abb. 5.5: Alle Klassen haben gleiche diagonale Kovarianzmatrizen aber ungleiche Varianzen.

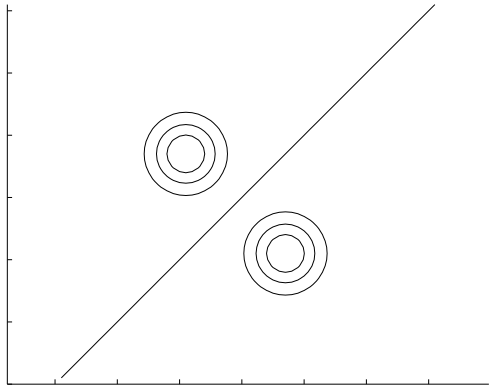


Abb. 5.6: Alle Klassen haben gleiche diagonale Kovarianzmatrizen mit gleicher Varianz in beiden Dimensionen.

Wenn die a-priori-Wahrscheinlichkeiten der Klassen gleich sind, ergibt sich $g_i(\boldsymbol{x}) = -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2$. Hierbei spricht man vom *Nächster-Mittelwert-Klassifikator* (engl. *nearest mean classifier*), da die Eingabewerte den Klassen des nächsten Mittelwertes zugewiesen werden. Wenn man jeden Mittelwert als idealen Prototyp oder als Muster für die Klasse annimmt, so ist dies eine Prozedur des *Musterabgleichs*. Dies kann erweitert werden zu

NÄCHSTER-MITTELWERT-KLASSIFIKATOR

MUSTERABGLEICH

$$\begin{aligned} g_i(\boldsymbol{x}) &= -\|\boldsymbol{x} - \boldsymbol{m}_i\|^2 = -(\boldsymbol{x} - \boldsymbol{m}_i)^T(\boldsymbol{x} - \boldsymbol{m}_i) \\ &= -(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{m}_i^T\boldsymbol{x} + \boldsymbol{m}_i^T\boldsymbol{m}_i)\,. \end{aligned} \tag{5.26}$$

Der erste Term $\boldsymbol{x}^T\boldsymbol{x}$ ist allen $g_i(\boldsymbol{x})$ gemein und kann weggelassen werden, so dass wir die Diskriminanzfunktion schreiben können als

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}\,, \tag{5.27}$$

wobei $\boldsymbol{w}_i = \boldsymbol{m}_i$ und $w_{i0} = -(1/2)\|\boldsymbol{m}_i\|^2$. Wenn alle $\boldsymbol{m}_i$ ähnliche Beträge haben, kann dieser Term ebenfalls ignoriert werden und wir nutzen

$$g_i(\boldsymbol{x}) = \boldsymbol{m}_i^T\boldsymbol{x}\,. \tag{5.28}$$

Wenn die Beträge von $\boldsymbol{m}_i$ vergleichbar sind, kann auch das Skalarprodukt als Ähnlichkeitsmaß genutzt werden statt der (negativen) Euklidischen Distanz.

Genaugenommen können wir uns das Finden der besten Diskriminanzfunktion als Aufgabe vorstellen, die beste Distanzfunktion zu finden. Dies kann als weiterer Ansatz zur Klassifikation betrachtet werden. Anstelle die Diskriminanzfunktionen $g_i(\boldsymbol{x})$ zu lernen, wollen wir lieber die geeignete Distanzfunktion $\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ lernen. Für beliebige $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, wobei $\boldsymbol{x}_1$ und $\boldsymbol{x}_2$ zur selben Klasse gehören und $\boldsymbol{x}_1$ sowie $\boldsymbol{x}_3$ zwei verschiedenen Klassen angehörig sind, besteht unser Ziel darin, Folgendes zu erhalten:

$$\mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_2) < \mathcal{D}(\boldsymbol{x}_1, \boldsymbol{x}_3)\,.$$

5.6 Anpassen der Komplexität

In Tabelle 5.1 sehen wir, wie die Anzahl von Parametern der Kovarianzmatrix reduziert werden kann, wobei der Komfort eines einfachen Modells gegen Generalität getauscht wird. Dies ist ein weiteres Beispiel für das Dilemma von Verzerrung/Varianz. Wenn wir vereinfachende Annahmen über die Kovarianzmatrix treffen und die Anzahl an zu berechnenden Parametern verringern, riskieren wir, dass Verzerrung eingebracht wird (siehe Abbildung 5.7). Andererseits aber, wenn keine solche Annahmen gemacht werden und die Matrizen willkürlich sind, kann die quadratische Diskriminanz unter Umständen große Varianz bei kleinen Datensätzen aufweisen. Der Idealfall hängt von der Komplexität des durch die vorliegenden Daten repräsentierten Problems und der Menge an verfügbaren Daten ab. Wenn uns nur ein kleiner Datensatz vorliegt, ist es möglicherweise besser – selbst wenn die Kovarianzmatrizen unterschiedlich sind – von einer gemeinsamen Kovarianzmatrix auszugehen; eine einzelne Kovarianzmatrix hat weniger Parameter und kann geschätzt werden, indem mehr Daten, das heißt Instanzen von Klassen, genutzt werden. Dieses Vorgehen ist äquivalent zur Nutzung von *linearen Diskriminanzfunktionen*, was bei der Klassifikation recht häufig geschieht und was wir ausführlicher in Kapitel 10 behandeln werden.

Man beachte, dass wir beim Nutzen der Euklidischen Distanz zur Ähnlichkeitsmessung annehmen, dass alle Variablen die selbe Varianz haben und unabhängig sind. In vielen Fällen trifft dies jedoch nicht zu; beispielsweise liegen Alter und Einkommen in unterschiedlichen Einheiten vor, und sind in vielerlei Kontexten abhängig voneinander. In so einem Fall können die Eingabewerte in einem Vorbearbeitungsschritt separat z-normalisiert werden (so dass sie einen Nullmittelwert und Einheitsvarianz haben), gefolgt von der Nutzung der Euklidischen Distanz. Andererseits ist es manchmal – selbst wenn die Variablen abhängig sind – doch besser anzunehmen, dass sie unabhängig sind und den naiven Bayesschen Klassifikator zu nutzen, wenn uns nicht genug Daten vorliegen, um die Abhängigkeit genau zu berechnen.

Regularized Discriminant Analysis

Friedman (1989) schlug eine Methode vor, welche alle die genannten Fälle als Spezialfälle kombiniert; dabei handelt es sich um die sogenannte *Regularized Discriminant Analysis* (RDA), also sinngemäß die regularisierte Diskriminanzanalyse. Wir erinnern uns, dass die Regularisierung den Ansätzen entspricht, bei denen man mit hoher Varianz beginnt und Einschränkungen in Richtung niedrigerer Varianz vornimmt, jedoch mit dem Risiko, die Verzerrung zu erhöhen. Im Falle der parametrischen Klassifikation mit Gaußschen Dichten können die Kovarianzmatrizen als gewichteter Durchschnitt der drei Spezialfälle geschrieben werden:

$$\mathbf{S}_i' = \alpha\sigma^2\mathbf{I} + \beta\mathbf{S} + (1-\alpha-\beta)\mathbf{S}_i \,. \tag{5.29}$$

Für $\alpha = \beta = 0$ führt dies auf einen quadratischen Klassifikator. Für $\alpha = 0$, $\beta = 1$ sind die Kovarianzmatrizen allen gemein und wir erhalten

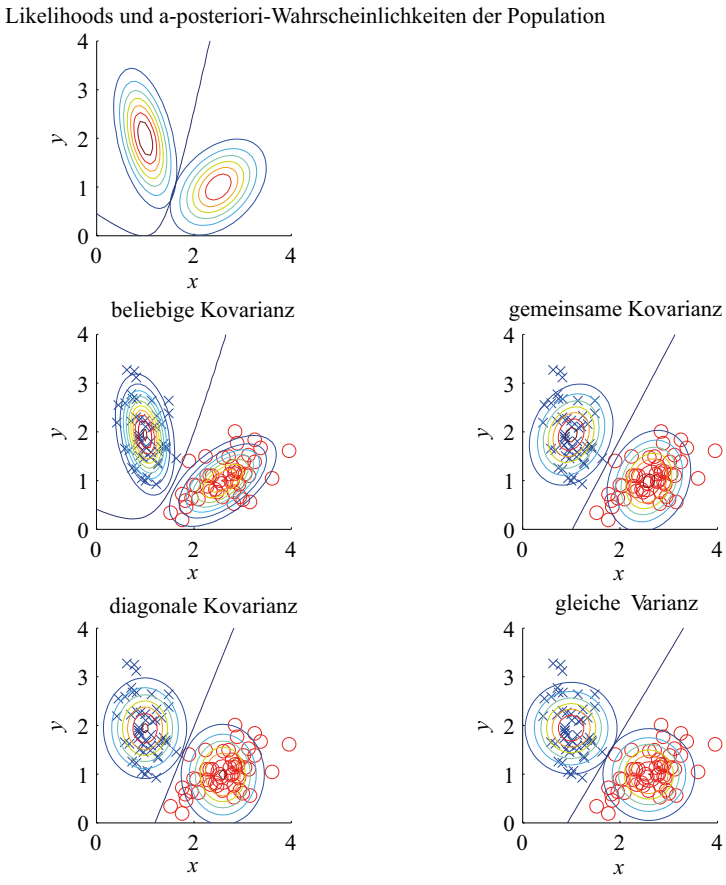


Abb. 5.7: Verschiedene Kovarianzmatrizen, die an die gleichen Daten angepasst sind, führen zu verschiedenen Grenzen.

lineare Klassifikatoren. Für $\alpha = 1$, $\beta = 0$ sind die Kovarianzmatrizen diagonal mit σ^2 auf der Diagonale, und wir erhalten den Nächsten-Mittelwert-Klassifikator. Zwischen diesen Extremen liegt eine Vielzahl von Klassifikatoren, und α, β werden durch Kreuzvalidierung optimiert.

Ein weiterer Ansatz zur Regularisierung bei kleinen Datensätzen besteht entweder in einem Bayesschen Ansatz, indem a-priori-Wahrscheinlichkeitsdichtefunktionen für $\boldsymbol{\mu}_i$ und $\mathbf{S}_i$ definiert werden, oder in einem Vorgehen, das die Kreuzvalidierung nutzt, um den besten der vier in Tabelle 5.1 gegebenen Fälle auszuwählen.

Tabelle 5.1: Varianzreduktion durch vereinfachende Annahmen.

Annahme	Kovarianzmatrix	Anzahl an Parametern
Geteilt, hypersphärisch	$\mathbf{S}_i = \mathbf{S} = s^2\mathbf{I}$	1
Geteilt, axial ausgerichtet	$\mathbf{S}_i = \mathbf{S}$, mit $s_{ij} = 0$	d
Geteilt, hyperelliptisch	$\mathbf{S}_i = \mathbf{S}$	$d(d+1)/2$
Verschieden, hyperelliptisch	$\mathbf{S}_i$	$K \cdot (d(d+1)/2)$

5.7 Diskrete Merkmale

In einigen Anwendungen haben wir diskrete Attribute, welche einen von n verschiedenen Werte annehmen. Mögliche Attribute sind beispielsweise die Farbe $\in$ {rot, blau, grün, schwarz} oder Pixel $\in$ {ein, aus}. Gehen wir davon aus, dass x_j binäre (Bernoulli)-Variablen sind, wobei

$$p_{ij} \equiv p(x_j = 1|\mathcal{C}_i) \,.$$

Wenn x_j unabhängige binäre Variablen sind, so haben wir

$$p(\boldsymbol{x}|\mathcal{C}_i) = \prod_{j=1}^{d} p_{ij}^{x_j} (1 - p_{ij})^{(1-x_j)} \,.$$

Dies ist ein weiteres Beispiel für den naiven Bayesschen Klassifikator, bei dem $p(x_j|\mathcal{C}_i)$ einer Bernoulli-Verteilung unterliegen. Die Diskriminanzfunktion lautet

$$\begin{aligned} g_i(\boldsymbol{x}) &= \log p(\boldsymbol{x}|\mathcal{C}_i) + \log P(\mathcal{C}_i) \\ &= \sum_j \left[x_j \log p_{ij} + (1 - x_j) \log(1 - p_{ij})\right] + \log P(\mathcal{C}_i) \end{aligned} \tag{5.30}$$

und ist linear. Der Schätzer für p_{ij} ist

$$\hat{p}_{ij} = \frac{\sum_t x_j^t r_i^t}{\sum_t r_i^t} \,. \tag{5.31}$$

Dokumentenkategorisierung

Bag of Words

Dieser Ansatz wird bei der *Dokumentenkategorisierung* genutzt, die beispielsweise bei der Einordnung von Nachrichten in verschiedene Kategorien wie Politik, Sport, Mode usw. Anwendung findet. Bei der *Bag-of-Words-Repräsentation* wählen wir a priori d Wörter, von denen wir glauben, dass sie Information im Hinblick auf die jeweilige Klasse liefern (Manning und Schütze 1999). Nützliche Wörter sind bei der Klassifikation

von Nachrichten zum Beispiel „Granate“, „Athlet“ oder „Kollektion“, nicht aber „Angriff“ oder „Trend“, weil diese beiden Wörter mehrdeutig sind. In der Bag-of-Words-Repräsentation ist jeder Text ein d-dimensionaler Vektor. Die Komponente x_j ist 1, wenn Wort j im Dokument vorkommt; anderenfalls ist sie 0. Bei dieser Repräsentation gehen alle Informationen über die Anordnung der Wörter verloren – daher die Bezeichnung *Bag* of Words (*Tasche* voller Wörter).

Nach dem Training schätzt $\hat{p}_{ij}$ die Wahrscheinlichkeit, dass Wort j im Dokumententyp i vorkommt. Wörter, deren Wahrscheinlichkeiten für verschiedene Klassen ähnlich sind, liefern nicht viel Information. Damit Wörter nützlich sind, sollte die Wahrscheinlichkeit für eine Klasse (oder für einige wenige) hoch sein und für alle anderen sehr klein. Wir werden über diese Form der *Merkmalsselektion* in Kapitel 6 diskutieren. Eine andere Beispielanwendung für das Kategorisieren von Dokumenten sind *Spamfilter*. Dabei gibt es zwei Klassen von E-Mails: Spam und legitime Mails. Auch in der Bioinformatik sind die Eingaben im Allgemeinen Sequenzen diskreter Einheiten, etwa von Basenpaaren oder Aminosäuren.

SPAMFILTER

Für den allgemeinen Fall wollen wir annehmen, dass anstelle binärer Merkmale multinomiale x_j gegeben sind, die aus der Menge $\{v_1, v_2, \ldots, v_{n_j}\}$ ausgewählt wurden. Wir definieren neue 0/1-Dummy-Variablen als

$$z_{jk}^t = \begin{cases} 1 \text{ falls } x_j^t = v_k \,, \\ 0 \text{ andernfalls} \,. \end{cases}$$

Sei p_{ijk} die Wahrscheinlichkeit dafür, dass x_j, zugehörig zur Klasse $\mathcal{C}_i$, den Wert v_k annimmt.

$$p_{ijk} \equiv p(z_{jk} = 1 | \mathcal{C}_i) = p(x_j = v_k | \mathcal{C}_i)$$

Wenn die Attribute unabhängig sind, erhalten wir

$$p(\boldsymbol{x}|\mathcal{C}_i) = \prod_{j=1}^{d} \prod_{k=1}^{n_j} p_{ijk}^{z_{jk}} \,. \tag{5.32}$$

Die Diskriminanzfunktion ist daher

$$g_i(\boldsymbol{x}) = \sum_j \sum_k z_{jk} \log p_{ijk} + \log P(\mathcal{C}_i) \,. \tag{5.33}$$

Der Maximum-Likelihood-Schätzer für p_{ijk} ist

$$\hat{p}_{ijk} = \frac{\sum_t z_{jk}^t r_i^t}{\sum_t r_i^t} \,, \tag{5.34}$$

was in Gleichung 5.33 eingesetzt werden kann, damit wir die Diskriminanzfunktion erhalten.

5.8 Multivariate Regression

MULTIVARIATE LINEARE REGRESSION

Bei der *multivariaten linearen Regression* wird angenommen, dass die numerische Ausgabe r als lineare Funktion vorliegt, das heißt als gewichtete Summe von verschiedenen Eingabevariablen $x_1, \ldots, x_d$ und Rauschen. In der Statistikliteratur spricht man eigentlich von der *multiplen Regression*; Statistiker nutzen den Begriff *multivariat* bei Vorhandensein von multiplen Ausgaben. Das multivariate lineare Modell ist

$$\begin{aligned} r^t &= g(\boldsymbol{x}^t | w_0, w_1, \ldots, w_d) + \epsilon \\ &= w_0 + w_1 x_1^t + w_2 x_2^t + \cdots + w_d x_d^t + \epsilon \,. \end{aligned} \tag{5.35}$$

Ebenso wie beim univariaten Fall nehmen wir ϵ als normalverteilt mit Nullmittelwert und konstanter Varianz an, und die Maximierung der Likelihood ist äquivalent zur Minimierung der Summe der quadrierten Fehler:

$$E(w_0, w_1, \ldots, w_d | \mathcal{X}) = \frac{1}{2} \sum_t (r^t - w_0 - w_1 x_1^t - w_2 x_2^t - \cdots - w_d x_d^t)^2 . \tag{5.36}$$

Bilden wir dann die Ableitung bezüglich der Parameter, $w_j, j = 0, \ldots, d$, so erhalten wir die *Normalengleichungen*:

$$\begin{aligned} \sum_t r^t &= N w_0 + w_1 \sum_t x_1^t + w_2 \sum_t x_2^t + \cdots + w_d \sum_t x_d^t \qquad (5.37) \\ \sum_t x_1^t r^t &= w_0 \sum_t x_1^t + w_1 \sum_t (x_1^t)^2 + w_2 \sum_t x_1^t x_2^t + \cdots + w_d \sum_t x_1^t x_d^t \\ \sum_t x_2^t r^t &= w_0 \sum_t x_2^t + w_1 \sum_t x_1^t x_2^t + w_2 \sum_t (x_2^t)^2 + \cdots + w_d \sum_t x_2^t x_d^t \\ &\vdots \\ \sum_t x_d^t r^t &= w_0 \sum_t x_d^t + w_1 \sum_t x_d^t x_1^t + w_2 \sum_t x_d^t x_2^t + \cdots + w_d \sum_t (x_d^t)^2 \,. \end{aligned}$$

Definieren wir nun die folgenden Vektoren und die Matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_d^2 \\ \vdots & & & & \\ 1 & x_1^N & x_2^N & \cdots & x_d^N \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d, \end{bmatrix}, \quad \boldsymbol{r} = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix},$$

dann können die Normalengleichungen notiert werden als

$$\mathbf{X}^T \mathbf{X} \boldsymbol{w} = \mathbf{X}^T \boldsymbol{r} \tag{5.38}$$

und wir können nach den Parametern auflösen und erhalten

$$\boldsymbol{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{r}\,. \tag{5.39}$$

Diese Methode ist identisch mit der, die wir für die polynomiale Regression verwendet haben, wobei eine Eingabe genutzt wird. Die zwei Problemstellungen sind gleich, wenn wir die Variablen als $x_1 = x, x_2 = x^2, \ldots, x_k = x^k$ definieren. Dies gibt uns auch einen Hinweis darauf, wie wir die *multivariate polynomiale Regression* durchführen können, sofern dies nötig ist (Übung 7). Außer bei kleinen d nutzen wir jedoch bei der multivariaten Regression kaum Polyome eines höheren Grades als des linearen.

MULTIVARIATE POLYNOMIALE REGRESSION

Tatsächlich ist die Verwendung von Termen höherer Ordnung als zusätzliche Eingaben nur eine Möglichkeit unter vielen. Wir können eine beliebige nichtlineare Funktion der ursprünglichen Eingaben mithilfe von Basisfunktionen definieren. Beispielsweise können wir Eingaben $x_2 = \sin(x)$, $x_3 = \exp(x^2)$ definieren, wenn wir glauben, dass eine solche Transformation sinnvoll ist. Die Verwendung eines linearen Modells in diesem neuen erweiterten Raum entspricht dann einem nichtlinearen Modell im ursprünglichen Raum. Die Rechnung bleibt weiterhin gültig, wir müssen nur $\mathbf{X}$ durch die Datenmatrix ersetzen, nachdem die Basisfunktionen angewendet wurden. Wie wir später in unterschiedlichen Zusammenhängen sehen werden (z. B. bei mehrlagigen Perzeptronen, Support-Vektor-Maschinen und Gaußschen Prozessen), wird dieser Weg, das lineare Modell zu verallgemeinern, sehr häufig genutzt.

Ein Vorteil linearer Modelle liegt darin, dass wir nach der Regression unter Betrachtung der $w_j, j = 1, \ldots, d$ Werte Wissen extrahieren können. Zunächst sehen wir anhand der Vorzeichen von w_j, ob x_j einen positiven oder negativen Effekt auf die Ausgabe hat. Wenn alle x_j im selben Bereich liegen, können wir zweitens durch Betrachtung der Absolutwerte von w_j eine Vorstellung davon bekommen, wie wichtig ein Merkmal ist, die Merkmale hinsichtlich ihrer Wichtigkeit sortieren, und sogar die Merkmale entfernen, deren w_j nahe Null liegen.

Bei Vorhandensein von multiplen Ausgaben können diese äquivalent als eine Menge von unabhängigen Regressionsproblemen mit jeweils eindimensionaler Ausgabe definiert werden.

5.9 Anmerkungen

Eine gute Übersicht über die lineare Algebra bietet Strang 2006. Harville 1997 ist ebenfalls ein exzellentes Buch, in dem die Matrixalgebra aus einer statistischen Perspektive betrachtet wird.

Eine Unannehmlichkeit bei multivariaten Daten besteht darin, dass bei einer großen Zahl von Dimensionen eine visuelle Analyse unmöglich

wird. Es gibt in der Statistikliteratur Vorschläge zu Methoden der visuellen Darstellung multivariater Daten; ein Überblick findet sich bei Rencher 1995. Eine Möglichkeit besteht darin, jeweils zwei mal zwei Variablen als bivariate Streudiagramme darzustellen. Sind die Daten multivariat normalverteilt, dann sollte das Diagramm von beliebigen zwei Variablen in etwa linear sein; dies kann als visueller Test auf multivariate Normalität hin genutzt werden. Eine andere Möglichkeit, die wir in Kapitel 6 diskutieren werden, ist, sie auf eine oder zwei Dimensionen zu projizieren und dort visuell darzustellen.

Ein großer Teil der Arbeiten zur Mustererkennung wurde unter der Annahme von multivariaten Normalverteilungen geleistet. Manchmal spricht man bei solch einer Diskriminanzfunktion sogar vom optimalen Bayesschen Klassifikator, jedoch ist dies im Allgemeinen falsch; sie ist nur optimal, wenn die Dichten tatsächlich multivariate Normalverteilungen sind und wir genügend Daten haben, um die korrekten Parameter anhand der Daten zu berechnen. Rencher 1995 diskutiert Tests zur Bewertung von multivariater Normalität sowie Tests zur Überprüfung auf gleiche Kovarianzmatrizen. McLachlan 1992 befasst sich mit der Klassifikation mit multivariaten Normalen und vergleicht lineare und quadratische Diskriminanzfunktionen.

Eine offensichtliche Einschränkung der multivariaten Normalen ist die, dass sie keine Daten zulassen, die einige diskrete Merkmale enthalten. Eine Variable mit n möglichen Werten kann in n 0/1-Dummy-Variablen überführt werden, jedoch erhöht dies die Dimensionalität. Man kann in diesem n-dimensionalen Raum eine Dimensionalitätsreduktion mittels einer der in Kapitel 6 erläuterten Methoden vornehmen und dadurch eine Erhöhung der Dimensionalität vermeiden. Die parametrische Klassifikation für solche Fälle gemischter Merkmale wird detailiert bei McLachlan 1992 behandelt.

5.10 Übungen

1. Beweisen Sie Gleichung 5.11.

   Lösung: Gegeben ist

   $$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

   und somit haben wir

   $$\begin{aligned} |\mathbf{\Sigma}| &= \sigma_1^2\sigma_2^2 - \rho^2\sigma_1^2\sigma_2^2 = \sigma_1^2\sigma_2^2(1-\rho^2)\,, \\ |\mathbf{\Sigma}|^{1/2} &= \sigma_1\sigma_2\sqrt{1-\rho^2}\,, \\ \mathbf{\Sigma}^{-1} &= \frac{1}{\sigma_1^2\sigma_2^2(1-\rho^2)} \begin{bmatrix} \sigma_2^2 & -\rho\sigma_1\sigma_2 \\ -\rho\sigma_1\sigma_2 & \sigma_1^2 \end{bmatrix}. \end{aligned}$$

$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ kann entwickelt werden in

$$\begin{bmatrix} x_1-\mu_1 & x_2-\mu_2 \end{bmatrix} \begin{bmatrix} \frac{\sigma_2^2}{\sigma_1^2\sigma_2^2(1-\rho^2)} & -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2^2(1-\rho^2)} \\ -\frac{\rho\sigma_1\sigma_2}{\sigma_1^2\sigma_2^2(1-\rho^2)} & \frac{\sigma_1^2}{\sigma_1^2\sigma_2^2(1-\rho^2)} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{bmatrix}$$
$$= \frac{1}{1-\rho^2}\left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - 2\rho\left(\frac{x_1-\mu_1}{\sigma_1}\right)\left(\frac{x_2-\mu_2}{\sigma_2}\right) + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right].$$

2. Generieren Sie eine Stichprobe aus einer multivariaten Normalverteilung $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, berechnen Sie $\boldsymbol{m}$ und $\mathbf{S}$, und vergleichen Sie sie mit $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Überprüfen Sie, wie Ihre Schätzungen sich mit wechselnder Stichprobengröße ändern.

3. Generieren Sie Stichproben aus zwei multivariaten Normalverteilungen $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, $i = 1, 2$ und berechnen Sie die optimale Bayessche Diskriminanzfunktion für die vier Fälle aus Tabelle 5.1.

4. Leiten Sie für ein Zweiklassenproblem für die vier Fälle der Gaußschen Dichten aus Tabelle 5.1 ab:
$$\log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{P(\mathcal{C}_2|\boldsymbol{x})}.$$

5. Eine andere Möglichkeit, Gaußsche Dichten zu verwenden, besteht darin, sie alle diagonal zu wählen, wobei sie jedoch alle unterschiedlich sein dürfen. Leiten Sie die Diskriminanzfunktion für diesen Fall her.

6. Angenommen, wir haben in zwei Dimensionen zwei Klassen mit genau dem gleichen Mittelwert. Welche Art von Grenzen können dann definiert werden?

7. Nehmen wir an, wir haben zwei Variablen x_1 und x_2, und wir wollen für diese eine quadratische Anpassung vornehmen, also
$$f(x_1, x_2) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4(x_1)^2 + w_5(x_2)^2.$$
Wie können wir $w_i, i = 0, \ldots, 5$ bei gegebener Stichprobe $\mathcal{X} = \{x_1^t, x_2^t, r^t\}$ bestimmen?
LÖSUNG: Wir schreiben diese Anpassung als
$$f(x_1, x_2) = w_0 + w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5$$
mit $z_1 = x_1$, $z_2 = x_2$, $z_3 = x_1x_2$, $z_4 = (x_1)^2$ und $z_5 = (x_2)^2$. Durch lineare Regression können wir $w_i, i = 0, \ldots, 5$ lernen. Die lineare Anpassung im fünfdimensionalen $(z_1, z_2, z_3, z_4, z_5)$-Raum entspricht einer quadratischen Anpassung im zweidimensionalen (x_1, x_2)-Raum. Wir diskutieren solche verallgemeinerten linearen Modelle (und weitere nichtlineare Basisfunktionen) ausführlicher in Kapitel 10.

8. Bei der Regression haben wir gesehen, dass die Anpassung eines quadratischen Modells äquivalent ist mit der Anpassung eines linearen Modells mit einer zusätzlichen Eingabe, die dem Quadrat der Eingabe entspricht. Ist dies auch bei der Klassifikation möglich?

   Lösung: Ja. Wir können Hilfsvariablen definieren, die den Potenz- und Kreuzprodukttermen entsprechen und dann ein lineares Modell verwenden. Wir können z. B. wie in Übung 7 $z_1 = x_1$, $z_2 = x_2$, $z_3 = x_1 x_2$, $z_4 = (x_1)^2$ und $z_5 = (x_2)^2$ definieren und dann ein lineares Modell verwenden, um $w_i, i = 1, \ldots 5$ zu lernen. Die Diskriminanz im fünfdimensionalen $(z_1, z_2, z_3, z_4, z_5)$-Raum entspricht einer quadratischen Diskriminanz im zweidimensionalen (x_1, x_2)-Raum.

9. Bei der Clusteranalyse von Dokumenten kann die Mehrdeutigkeit von Wörtern reduziert werden, indem der Kontext berücksichtigt wird. Dies ist zum Beispiel dadurch möglich, dass Wortpaare betrachtet werden, etwa „Angriff“ in Kombination mit „Granate und demgegenüber „Angriff in Kombination mit „Pass“. Diskutieren Sie, wie dies implementiert werden kann.

5.11 Literaturangaben

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Friedman, J. H. 1989. „Regularized Discriminant Analysis.“ *Journal of American Statistical Association* 84: 165–175.

Harville, D. A. 1997. *Matrix Algebra from a Statistician's Perspective.* New York: Springer.

Manning, C. D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing.* Cambridge, MA: MIT Press.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Strang, G. 2006. *Linear Algebra and its Applications,* 4th ed. Boston: Cengage Learning.

6 Dimensionalitätsreduktion

Die Komplexität eines beliebigen Klassifikators oder Regressors hängt von der Anzahl an Eingaben ab. Dadurch sind in gewisser Weise sowohl die Laufzeit- als auch die Speicherkomplexität sowie die Anzahl an Beispielen, um solch einen Klassifikator oder Regressor zu trainieren, festgelegt. In diesem Kapitel diskutieren wir Methoden der Merkmalsselektion, die eine Teilmenge wichtiger Merkmale auswählen und die restlichen Merkmale verwerfen. Außerdem befassen wir uns mit Methoden der Merkmalsextraktion, bei denen einige wenige neue Merkmale aus den ursprünglichen Eingaben herausgezogen werden.

6.1 Einführung

Bei einer Anwendung – gleich ob es sich dabei um eine Klassifikation oder Regression handelt – werden Beobachtungsdaten, von denen wir annehmen, dass sie Informationen enthalten, als Eingabe genutzt und in das System zu Zwecken der Entscheidungsfindung eingespeist. Idealerweise sollten wir die Merkmalsauswahl oder -extraktion nicht extra als separaten Prozess benötigen; der Klassifikator (oder Regressor) sollte in der Lage sein, die jeweils nötigen Merkmale zu verwenden und die irrelevanten zu verwerfen. Allerdings gibt es mehrere Gründe, warum wir an der Reduktion der Dimensionalität als eigenständigem Vorbearbeitungsschritt interessiert sind:

- Bei den meisten Lernalgorithmen hängt die Komplexität von der Anzahl an Eingabedimensionen d sowie von der Größe der Datenstichprobe N ab und aus Gründen der Reduzierung von Speicherplatz und Rechenaufwand interessiert uns eine Reduktion der Dimensionalität des Problems. Durch die Verkleinerung von d wird auch die Komplexität des Inferenzalgorithmus während der Testphase verringert.

- Wenn eine Eingabe als unnötig eingestuft wird, ersparen wir uns den Aufwand, sie zu extrahieren.

- Einfachere Modelle sind robuster bei kleineren Datenmengen. Einfachere Modelle haben weniger Varianz, das heißt, sie variieren

weniger stark in Abhängigkeit von den Eigenheiten einer Stichprobe, inklusive Rauschen, Ausreißer, und so weiter.

- Wenn Daten mit weniger Merkmalen erklärt werden können, erhalten wir eine bessere Vorstellung von dem den Daten zugrundeliegenden Prozess, wodurch Wissensextraktion möglich wird. Diese wenigen Merkmale können als *versteckte* oder *latente Faktoren* interpretiert werden, die in ihrem Zusammenspiel die beobachteten Merkmale generieren.
- Wenn Daten in einigen wenigen Dimensionen ohne Informationsverlust repräsentiert werden, so können sie graphisch dargestellt und visuell auf ihre Struktur und Ausreißer hin analysiert werden.

Es gibt zwei hauptsächliche Methoden zur Dimensionalitätsreduktion: die Merkmalsselektion und die Merkmalsextraktion. Bei der *Merkmalsselektion* sind wir darum bemüht, die k Vertreter aus den d Dimensionen herauszufinden, die uns die meisten Informationen liefern; wir verwerfen die anderen $(d-k)$ Dimensionen. Wir werden die *Teilmengenselektion* als eine Methode der Merkmalsselektion behandeln.

MERKMALSSELEKTION

Bei der *Merkmalsextraktion* sind wir daran interessiert, eine neue Menge von k Dimensionen zu finden, welche eine Kombination der originalen d Dimensionen sind. Diese Methoden können überwacht sein oder nicht, je nachdem, ob sie die Ausgabeinformationen nutzen oder nicht. Die bekanntesten und am meisten genutzten Methoden zur Merkmalsextraktion sind die *Hauptkomponentenanalyse* und die *lineare Diskriminanzanalyse*; beides sind lineare Projektionsmethoden, wobei erstere nicht überwacht und letztere überwacht ist. Die Hauptkomponentenanalyse ähnelt zwei anderen nichtüberwachten linearen Projektionsmethoden, die wir ebenfalls diskutieren werden, nämlich der *Faktorenanalyse* und der *multidimensionalen Skalierung*. Wenn wir nicht nur eine, sondern zwei Sätze von beobachteten Variablen haben, dann kann die *kanonische Korrelationsanalyse* verwendet werden, um die gemeinsamen Merkmale zu finden, welche die Abhängigkeit zwischen den beiden Variablensätzen erklären. Als Beispiele für Verfahren der nichtlinearen Dimensionalitätsreduktion betrachten wir das *isometrische Merkmalsmapping,* die *lokal lineare Einbettung* und *Laplacesche Eigenmaps.*

MERKMALSEXTRAKTION

6.2 Teilmengenselektion

Bei der *Teilmengenselektion* interessiert uns die beste Teilmenge aus der Menge aller Merkmale. Diese beste Teilmenge enthält die geringste Anzahl an Dimensionen, die am meisten zur Genauigkeit beitragen. Wir verwerfen die verbleibenden unwichtigen Dimensionen. Mit Hilfe einer geeigneten Fehlerfunktion kann dieser Ansatz sowohl bei Problemen der Klassifikation als auch der Regression eingesetzt werden. Es gibt 2^d mögliche Teilmengen

TEILMENGENSELEKTION

für d Variablen, jedoch können wir nicht auf alle testen – es sei denn, d ist klein – und wir nutzen daher heuristische Methoden, um eine akzeptable (aber nicht optimale) Lösung in akzeptabler (polynomialer) Rechenzeit zu finden.

Vorwärtsselektion

Rückwärtsselektion

Es existieren zwei Ansätze. Bei der *Vorwärtsselektion* beginnen wir ohne Variablen und fügen diese dann eine nach der anderen hinzu, wobei bei jedem Schritt diejenige Variable hinzugefügt wird, welche den Fehler am meisten verringert, und zwar solange, bis das Hinzufügen weiterer Variablen keine (oder nur sehr geringe) Verringerungen des Fehlers mehr bewirkt. Bei der *Rückwärtsselektion* haben wir zu Beginn alle Variablen zur Verfügung und entfernen dann eine nach der anderen, wobei jedes Mal diejenige Variable entnommen wird, welche den Fehler am meisten verringert (oder ihn nur minimal erhöht), solange bis jede weitere Entfernung den Fehler signifikant erhöhen würde. Bei beiden Fällen sollte die Fehlerüberprüfung anhand eines Validierungsdatensatzes vorgenommen werden, welcher sich vom Trainingsdatensatz unterscheidet, da wir die Genauigkeit bei der Generalisierung testen wollen. Mit mehr Merkmalen erhalten wir in der Regel einen niedrigeren Trainingsfehler, aber nicht zwingend einen geringeren Validierungsfehler.

Sei F eine Merkmalsmenge von Eingabedimensionen, $x_i, i = 1, \ldots, d$. $E(F)$ sei der am Validierungsdatensatz hervorgerufene Fehler unter ausschließlicher Nutzung der Eingaben aus F. Je nach Anwendung ist der Fehler entweder der mittlere quadratische Fehler oder der Fehlklassifikationsfehler.

Bei der *sequentiellen Vorwärtsselektion* beginnen wir ohne Merkmale: $F = \varnothing$. Bei jedem Schritt und für alle möglichen x_i trainieren wir unser Modell auf der Trainingsmenge und berechnen $E(F \cup x_i)$ auf der Validierungsmenge. Dann wählen wir die Eingabe x_j, welche den geringsten Fehler verursacht,

$$j = \underset{i}{\operatorname{argmin}}\, E(F \cup x_i)\,, \tag{6.1}$$

und befolgen:

$$\text{addiere } x_j \text{ zu } F, \text{ falls } E(F \cup x_j) < E(F)\,. \tag{6.2}$$

Wir beenden die Prozedur, wenn das Hinzufügen weiterer beliebiger Merkmale E nicht mehr verringert. Unter Umständen können wir das Ganze auch schon eher abbrechen, wenn die Verringerung des Fehlers zu klein ist – vorrausgesetzt, es existiert ein nutzerdefinierter Schwellwert, der von den jeweiligen Anwendungsbedingungen abhängt – so dass wir also einen Kompromiss zwischen der Wichtigkeit des Fehlers und der Komplexität schließen. Das Hinzufügen eines weiteren Merkmals bringt Kosten für die Auswahl des Merkmals mit sich und verkompliziert außerdem den Klassifikator/Regressor.

WRAPPER

Dieser Algorithmus ist auch als *Wrapper-Ansatz* bekannt. Hinter der Bezeichnung Wrapper (engl. für Hülle) steht die Vorstellung, dass der Prozess der Merkmalsextraktion den Lerner „umhüllt“, den er als Subroutine benutzt (Kohavi und John 2007).

Betrachten wir ein Beispiel für Irisdaten aus dem UCI-Repository, das vier Eingaben und drei Klassen hat. Es gibt fünfzig Instanzen pro Klasse, von denen wir zwanzig für das Training und die restlichen dreißig für die Validierung verwenden. Als Klassifikator verwenden wir den Nächster-Mittelwert-Klassifikator (siehe Gleichung 5.26 in Abschnitt 5.5). Wir beginnen mit einem einzelnen Merkmal; Abbildung 6.1 zeigt die grafischen Darstellungen der Trainingsdaten bei Verwendung von Einzelmerkmalen. Wird in diesen vier eindimensionalen Merkmalsräumen der Nächster-Mittelwert-Klassifikator verwendet, dann ergeben sich Validierungsgenauigkeiten von 0,76, 0,57, 0,92 und 0,94. Folglich wählen wir das vierte Merkmal (F4) als unser erstes Merkmal. Dann prüfen wir, ob das Hinzufügen eines weiteren Merkmals zu einer Verbesserung führt. Die bivariaten Darstellungen sind in Abbildung 6.2 gezeigt. Die zugehörigen Validierungsgenauigkeiten bei Verwendung des Nächster-Mittelwert-Klassifikators in diesen zweidimensionalen Räumen sind 0,87,

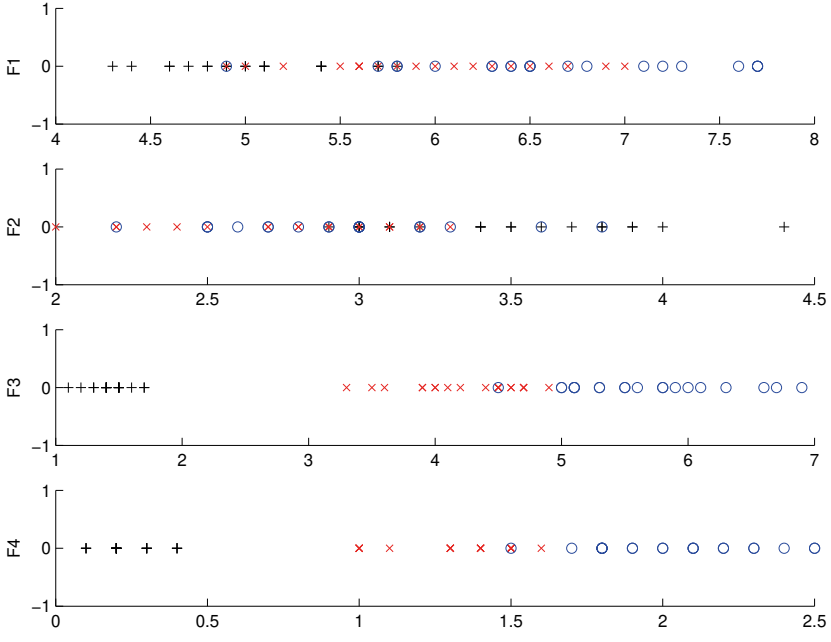


***Abb. 6.1*:** Grafische Darstellung der Trainingsdaten für Einzelmerkmale von Irisdaten. Die drei Klassen sind durch unterschiedliche Symbole gekennzeichnet. Wie man hier sieht, erlaubt das Merkmal F4 allein eine recht gute Unterscheidung.

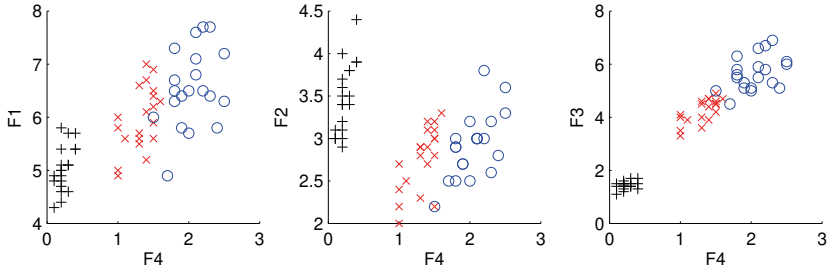


***Abb. 6.2*:** Grafische Darstellung der Trainingsdaten, wenn F4 zusammen mit einem der Merkmale F1, F2 oder F3 gewählt wird. Die beste Separation wird für (F3, F4) erreicht.

0,92 und 0,96 für (F1, F4), (F2, F4) und (F3, F4). Folglich fügen wir das dritte Merkmal als unser zweites Merkmal hinzu. Dann prüfen wir, ob das Hinzufügen des ersten oder zweiten Merkmals zu einer weiteren Verbesserung führt. Die Validierungsgenauigkeiten bei Verwendung des Nächster-Mittelwert-Klassifikators in diesen dreidimensionalen Räumen sind beide 0,94, so dass wir das dritte und vierte Merkmal zu unseren ausgewählten Merkmalen hinzufügen und aufhören. Im Übrigen erhalten wir eine Validierungsgenauigkeit von 0,94, wenn wir *alle vier* Merkmale verwenden – das Weglassen der beiden ersten führt zu einem Anstieg der Genauigkeit.

Man beachte, dass es stark vom verwendeten Klassifikator abhängt, welche Merkmale wir am Ende auswählen. Ein anderer wichtiger Punkt ist der, dass die ausgewählten Merkmale bei kleinen Datenmengen auch von der Aufteilung in Trainingsdaten und Validierungsdaten abhängen können. Deshalb kann es bei kleinen Datenmengen eine gute Idee sein, mehrere Unterteilungen in Traings- und Validierungsdaten vorzunehmen und die Entscheidung anhand der mittleren Validierungsleistung zu treffen. Solche Resampling-Verfahren werden wir in Kapitel 19 besprechen.

Dieser Prozess, bei dem ein Merkmal nach dem anderen getestet wird, kann kostspielig sein, da wir zur Verringerung der Dimensionen von d auf k das System $d + (d-1) + (d-2) + \cdots + (d-k)$ Mal trainieren und testen müssen; dies entspricht $O(d^2)$. Dabei handelt es sich um eine lokale Suchprozedur und es gibt keine Garantie, dass die optimale Teilmenge, sprich die minimale Teilmenge, die den geringsten Fehler verursacht, tatsächlich gefunden werden kann. Beispielsweise sind x_i und x_j allein möglicherweise nicht gut genug, aber zusammen verringern sie den Fehler vielleicht recht deutlich. Da es sich aber um einen Greedy-Algorithmus handelt und Attribute eins nach dem anderen hinzugefügt werden, ist er unter Umständen nicht in der Lage, diese Eigenheit zu entdecken. Es ist möglich zu generalisieren und m Merkmale statt einem auf einen Schlag hinzuzufügen; dies geht auf Kosten eines höheren Rechenaufwands. Wir

können den Prozess auch zurückverfolgen (backtracking) und überprüfen, welches vorher hinzugefügte Merkmal nach einer aktuellen Hinzufügung entfernt werden kann. Dadurch vergrößert sich zwar der Suchraum aber auch die Komplexität. Bei *flexiblen Suchmethoden* (Pudil, Novovičová und Kittler 1994) darf sich die Zahl der hinzugefügten und entfernten Merkmale auch bei jedem Schritt ändern.

FLEXIBLE SUCHMETHODEN

Bei der *sequentiellen Rückwärtsselektion* beginnen wir mit F, worin alle Merkmale enthalten sind, und vollziehen eine ähnliche Prozedur, abgesehen davon, dass wir ein Attribut von F entfernen, statt eines hinzuzufügen, und zwar entfernen wir dasjenige, welches nach seiner Entfernung den geringsten Fehler verursacht:

$$j = \underset{i}{\operatorname{argmin}} E(F - x_i) \tag{6.3}$$

und dann:

$$\text{entferne } x_j \text{ aus } F, \text{ wenn } E(F - x_j) < E(F)\,. \tag{6.4}$$

Wir beenden den Vorgang, wenn das Entfernen weiterer Merkmale den Fehler nicht mehr verringert. Um die Komplexität zu senken, können wir unter Umständen ein Merkmal auch dann entfernen, wenn dies den Fehler geringfügig vergrößert.

Alle möglichen Varianten für die Vorwärtssuche sind ebenfalls bei der Rückwärtssuche anwendbar. Die Komplexität der Rückwärtssuche weist dieselbe Größenordnung wie die der Vorwärtssuche auf, außer, dass das Testen eines Systems mit mehr Merkmalen aufwendiger ist als das eines Systems mit weniger Merkmalen. Außerdem ist die Vorwärtssuche vor allem dann vorzuziehen, wenn wir viele nutzlose Merkmale erwarten.

Die Teilmengenselektion ist insofern überwacht, als dass Ausgaben vom Regressor bzw. Klassifikator zur Berechnung des Fehlers genutzt werden, jedoch kann sie bei jeder beliebigen Regressions- oder Klassifikationsmethode angewendet werden. Im speziellen Fall der Klassifikation, in dem die original d-dimensionalen Klassendichten multivariaten Normalverteilungen gehorchen, ist jede Teilmenge der Klassenmerkmale ebenfalls multivariat normalverteilt. Daher kann auch für die Reduktion auf eine Teilmenge der Merkmale der parametrsiche Klassifkationsansatz angewendet werden mit dem Vorteil, dass statt einer $d \times d$ nur noch eine $k \times k$ Kovarianzmatrix geschätzt werden muss.

Bei einer Anwendung wie der Gesichtserkennung ist die Merkmalsselektion keine gute Methode zur Dimensionalitätsreduktion, da individuelle Pixel an sich kaum entscheidende Informationen enthalten; erst Kombinationen der Werte mehrerer Pixel zusammen tragen Informationen zur Identität des Gesichts bei. Dies wird durch Merkmalsextraktionsmethoden gelöst, mit denen wir uns als nächstes befassen.

6.3 Hauptkomponentenanalyse

Bei den Projektionsmethoden sind wir daran interessiert, eine Abbildung von den Eingaben im originalen d-dimensionalen Raum auf einen neuen $(k < d)$-dimensionalen Raum mit minimalem Informationsverlust zu finden. Die Projektion von $\boldsymbol{x}$ in Richtung von $\boldsymbol{w}$ lautet

$$z = \boldsymbol{w}^T \boldsymbol{x} \,. \tag{6.5}$$

HAUPTKOMPONENTENANALYSE

Die *Hauptkomponentenanalyse* (PCA, Principal Components Analysis) ist insofern eine unüberwachte Methode, als dass sie die Ausgabeinformationen nicht nutzt; das zu maximierende Kriterium ist die Varianz. Die Hauptkomponente ist $\boldsymbol{w}_1$, so dass eine Stichprobe nach der Projektion auf $\boldsymbol{w}_1$ so weit wie möglich ausgebreitet ist, so dass die Differenz zwischen den Stichprobenpunkten am offensichtlichsten wird. Für eine eindeutige Lösung, und auch, um die Richtung als den entscheidenden Faktor festzulegen, ist es nötig, dass $\|\boldsymbol{w}_1\| = 1$. Aus Gleichung 5.14 wissen wir, dass bei Gültigkeit von $z_1 = \boldsymbol{w}_1^T \boldsymbol{x}$ mit $\mathrm{Cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$ gilt:

$$\mathrm{Var}(z_1) = \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 \,.$$

Wir suchen nach $\boldsymbol{w}_1$, das $\mathrm{Var}(z_1)$ maximiert, unter der Bedingung, dass $\boldsymbol{w}_1^T \boldsymbol{w}_1 = 1$. Wenn wir dies als ein Lagrange-Problem schreiben, erhalten wir:

$$\max_{\boldsymbol{w}_1} \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 - \alpha(\boldsymbol{w}_1^T \boldsymbol{w}_1 - 1) \,. \tag{6.6}$$

Bilden wir die Ableitung hinsichtlich $\boldsymbol{w}_1$ und setzen das Ganze gleich null, bekommen wir

$$2\boldsymbol{\Sigma} \boldsymbol{w}_1 - 2\alpha \boldsymbol{w}_1 = 0, \text{ und somit } \boldsymbol{\Sigma} \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1 \,,$$

was erfüllt ist, wenn $\boldsymbol{w}_1$ ein Eigenvektor von $\boldsymbol{\Sigma}$ mit α als zugehörigem Eigenwert ist. Da wir

$$\boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 = \alpha \boldsymbol{w}_1^T \boldsymbol{w}_1 = \alpha$$

maximieren wollen, wählen wir den Eigenvektor mit dem größten Eigenwert, so dass die Varianz maximiert wird. Somit ist die Hauptkomponente der Eigenvektor der Kovarianzmatrix der Eingabestichprobe mit dem größten Eigenwert, $\lambda_1 = \alpha$.

Die zweite Hauptkomponente $\boldsymbol{w}_2$ sollte die Varianz ebenfalls maximieren, Einheitslänge aufweisen und orthogonal zu $\boldsymbol{w}_1$ liegen. Letztere Bedingung ergibt sich daraus, dass nach der Projektion $z_2 = \boldsymbol{w}_2^T \boldsymbol{x}$ unkorreliert zu z_1 sein soll. Für die zweite Hauptkomponente erhalten wir:

$$\max_{\boldsymbol{w}_2} \boldsymbol{w}_2^T \boldsymbol{\Sigma} \boldsymbol{w}_2 - \alpha(\boldsymbol{w}_2^T \boldsymbol{w}_2 - 1) - \beta(\boldsymbol{w}_2^T \boldsymbol{w}_1 - 0) \,. \tag{6.7}$$

Bilden wir die Ableitung hinsichtlich $\boldsymbol{w}_2$ und setzen das Ganze gleich null, erhalten wir

$$2\boldsymbol{\Sigma}\boldsymbol{w}_2 - 2\alpha\boldsymbol{w}_2 - \beta\boldsymbol{w}_1 = 0\,. \tag{6.8}$$

Das Multiplizieren beider Seiten von links mit $\boldsymbol{w}_1^T$ ergibt

$$2\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 - 2\alpha\boldsymbol{w}_1^T\boldsymbol{w}_2 - \beta\boldsymbol{w}_1^T\boldsymbol{w}_1 = 0\,.$$

Man beachte, dass $\boldsymbol{w}_1^T\boldsymbol{w}_2 = 0$. $\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2$ ist ein Skalar, gleich seiner Transponierten $\boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1$, wobei $\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_1$, da $\boldsymbol{w}_1$ der führende Eigenvektor von $\boldsymbol{\Sigma}$ ist. Somit gilt:

$$\boldsymbol{w}_1^T\boldsymbol{\Sigma}\boldsymbol{w}_2 = \boldsymbol{w}_2^T\boldsymbol{\Sigma}\boldsymbol{w}_1 = \lambda_1\boldsymbol{w}_2^T\boldsymbol{w}_1 = 0\,.$$

Dann ist $\beta = 0$ und Gleichung 6.8 reduziert sich auf

$$\boldsymbol{\Sigma}\boldsymbol{w}_2 = \alpha\boldsymbol{w}_2\,,$$

was impliziert, dass $\boldsymbol{w}_2$ der Eigenvektor von $\boldsymbol{\Sigma}$ mit dem zweitgrößten Eigenwert sein sollte, $\lambda_2 = \alpha$. Ähnlich hierzu können wir zeigen, dass die anderen Dimensionen durch die Eigenvektoren mit abnehmenden Eigenwerten gegeben sind.

Weil $\boldsymbol{\Sigma}$ symmetrisch ist, liegen die Eigenvektoren für zwei verschiedene Eigenwerte orthogonal zueinander. Wenn $\boldsymbol{\Sigma}$ positiv definit ist ($\boldsymbol{x}^T\boldsymbol{\Sigma}\boldsymbol{x} > 0$, für alle $\boldsymbol{x}$ ungleich null), sind alle zugehörigen Eigenwerte positiv. Ist $\boldsymbol{\Sigma}$ singulär, so ist der zugehörige Rang, also die effektive Dimensionalität, gleich k mit $k < d$, und $\lambda_i, i = k+1, \ldots, d$ sind gleich null (λ_i sind in absteigender Ordnung sortiert). Die k Eigenvektoren mit Eigenwerten ungleich null sind die Dimensionen des reduzierten Raumes. Der erste Eigenvektor (derjenige mit dem größten Eigenwert) $\boldsymbol{w}_1$, sprich die Hauptkomponente, erklärt den größten Teil der Varianz; der zweite erklärt den zweitgrößten Teil, und so fort.

Wir definieren

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{m})\,, \tag{6.9}$$

wobei die k Spalten von $\mathbf{W}$ die k führenden Eigenvektoren von $\mathbf{S}$ sind, dem Schätzer von $\boldsymbol{\Sigma}$. Wir subtrahieren den Stichprobenmittelwert $\boldsymbol{m}$ von $\boldsymbol{x}$ vor der Projektion, um die Daten um den Ursprung herum zu zentrieren. Nach dieser linearen Transformation erhalten wir einen k-dimensionalen Raum, dessen Dimensionen den Eigenvektoren entsprechen und bei dem die Varianzen über diese neuen Dimensionen gleich den Eigenwerten sind (siehe Abbildung 6.3). Um die Varianzen zu normalisieren, können wir durch die Quadratwurzeln der Eigenwerte dividieren.

Betrachten wir eine weitere Ableitung. Wir wollen die Matrix $\mathbf{W}$ finden, so dass wir, wenn wir $\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}$ haben (es sei ohne Informationsverlust

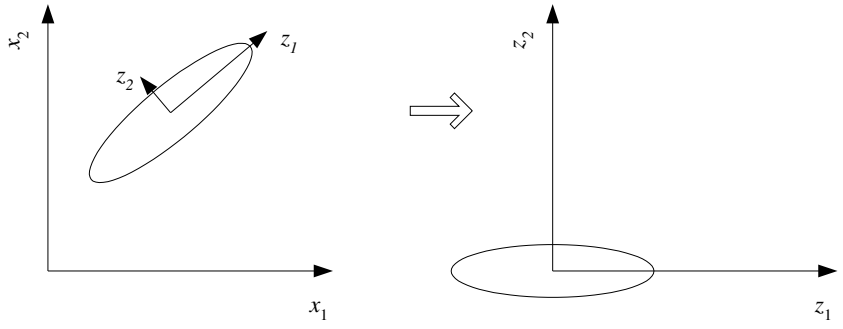


***Abb. 6.3*:** Die Hauptkomponentenanalyse zentriert die Stichprobe und rotiert dann die Achsen so, dass sie an den Ausrichtungen der höchsten Varianz orientiert sind. Wenn die Varianz für z_2 zu klein ist, kann sie ignoriert werden und wir erhalten eine Dimensionalitätsreduktion von zwei auf eins.

angenommen, dass die Werte für $\boldsymbol{x}$ bereits zentriert sind), $\text{Cov}(\boldsymbol{z}) = \mathbf{D}'$ erhalten, wobei $\mathbf{D}'$ eine beliebige diagonale Matrix ist, das heißt, wir suchen also nach unkorrelierten z_i.

Wenn wir eine $(d \times d)$ Matrix $\mathbf{C}$ bilden, deren i-te Spalte dem normalisierten Eigenvektor $\boldsymbol{c}_i$ von $\mathbf{S}$ entspricht, so ist $\mathbf{C}^T\mathbf{C} = \mathbf{I}$ und

$$\begin{aligned}
\mathbf{S} &= \mathbf{S}\mathbf{C}\mathbf{C}^T \\
&= \mathbf{S}(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_d)\mathbf{C}^T \\
&= (\mathbf{S}\boldsymbol{c}_1, \mathbf{S}\boldsymbol{c}_2, \ldots, \mathbf{S}\boldsymbol{c}_d)C^T \\
&= (\lambda_1\boldsymbol{c}_1, \lambda_2\boldsymbol{c}_2, \ldots, \lambda_d\boldsymbol{c}_d)\mathbf{C}^T \\
&= \lambda_1\boldsymbol{c}_1\boldsymbol{c}_1^T + \cdots + \lambda_d\boldsymbol{c}_d\boldsymbol{c}_d^T \\
&= \mathbf{C}\mathbf{D}\mathbf{C}^T \,. \qquad (6.10)
\end{aligned}$$

Dabei ist $\mathbf{D}$ eine diagonale Matrix, deren diagonale Elemente den Eigenwerten $\lambda_1, \ldots, \lambda_d$ entsprechen. Hierbei spricht man von der *Spektralzerlegung* von $\mathbf{S}$. Da $\mathbf{C}$ orthogonal ist und $\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{I}$, können wir auf der linken Seite mit $\mathbf{C}^T$ multiplizieren und auf der rechten mit $\mathbf{C}$, und erhalten

SPEKTRAL-ZERLEGUNG

$$\mathbf{C}^T\mathbf{S}\mathbf{C} = \mathbf{D} \,. \qquad (6.11)$$

Wenn $\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}$, so wissen wir, dass $\text{Cov}(\boldsymbol{z}) = \mathbf{W}^T\mathbf{S}\mathbf{W}$, was unserem Wunsch nach einer diagonalen Matrix entsprechen sollte. Ausgehend von Gleichung 6.11 erkennen wir, dass wir $\mathbf{W} = \mathbf{C}$ setzen können.

Betrachten wir ein Beispiel, um ein Gefühl für das Prozedere zu bekommen (Rencher 1995). Nehmen wir an, gegeben sei eine Klasse von Studenten mit Noten für fünf Kurse, und wir möchten diese Studenten ordnen. Das

heißt, wir wollen die Daten auf eine Dimension projizieren, so dass die Differenz zwischen den Datenpunkten am deutlichsten wird. Wir können die PCA nutzen. Der Eigenvektor mit dem höchsten Eigenwert ist die Ausrichtung mit der höchsten Varianz, das heißt, die Richtung, in welche die Studentendaten am meisten ausgebreitet sind. Dies funktioniert besser, als wenn wir mit dem Durchschnitt arbeiten würden, da wir Korrelationen und Differenzen zwischen Varianzen mit in die Betrachtungen einbeziehen.

In der Praxis – selbst wenn alle Eigenwerte größer als Null sind – sollte bei kleinem $|\mathbf{S}|$ und in Erinnerung der Tatsache, dass $|\mathbf{S}| = \prod_{i=1}^{d} \lambda_i$, klar sein, dass manche Eigenwerte kaum etwas zur Varianz beitragen und verworfen werden können. Dann betrachten wir die führenden k Komponenten, die mehr als, sagen wir 90% der Varianz erklären. Wenn λ_i in absteigender Ordnung sortiert sind, ergibt sich das durch die k Hauptkomponenten erklärte *Verhältnis der Varianz* als

Verhältnis der Varianz

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}.$$

Wenn die Dimensionen hochkorreliert sind, wird es eine geringe Anzahl an Eigenvektoren mit hohen Eigenwerten geben und k wird deutlich kleiner als d sein; eine starke Reduktion in der Dimensionalität kann unter Umständen erreicht werden. Dies ist typischerweise bei vielen Bild- und Sprachbearbeitungsprozessen der Fall, bei denen naheliegende Eingabewerte (in Raum oder Zeit) hochkorreliert sind. Wenn die Dimensionen nicht korrelieren, so ist k so groß wie d und es kann kein Gewinn durch die PCA erzielt werden.

Scree-Graph

Ein *Scree-Graph* ist die graphische Darstellung der Varianz, erklärt als eine Funktion der Anzahl an behaltenen Eigenvektoren (siehe Abbildung 6.4). Mit Hilfe einer visuellen Analyse kann k ebenfalls festgelegt werden. Am „Ellenbogen“ wird die erklärte Varianz durch das Hinzufügen eines weiteren Eigenvektors nicht signifikant erhöht.

Eine weitere Möglichkeit besteht darin, diejenigen Eigenvektoren zu ignorieren, deren Eigenwerte niedriger als die durchschnittliche Eingabevarianz sind. Da $\sum_i \lambda_i = \sum_i s_i^2$ (gleich der *Spur* von $\mathbf{S}$, bezeichnet durch $\text{tr}(\mathbf{S})$), ist der durchschnittliche Eigenwert gleich der durchschnittlichen Eingabevarianz. Wenn wir nur die Eigenvektoren mit Eigenwerten größer als der Durchschnittseigenwert behalten, umfasst das nur jene, die eine höhere Varianz haben als die durchschnittliche Eingabevarianz.

Wenn die Varianzen der originalen x_i Dimensionen deutlich variieren, beeinflussen sie die Richtung der Hauptkomponenten mehr als die Korrelationen, weshalb ein übliches Vorgehen darin besteht, die Daten vorzubearbeiten, so dass jede Dimensionen einen Nullmittelwert und eine Einheitsvarianz besitzt; erst dann findet die PCA statt. Oder man nutzt die Eigenvektoren der Korrelationsmatrix $\mathbf{R}$ anstelle der Kovarianzmatrix $\mathbf{S}$, damit die Korrelationen wirksam sind und nicht die individuellen Varianzen.

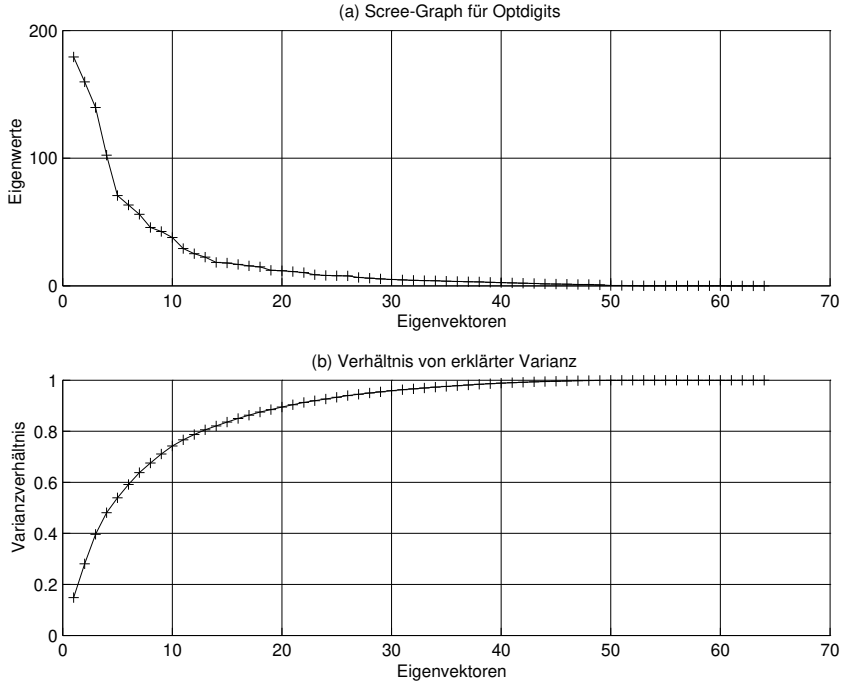


***Abb. 6.4*:** (a) Scree Graph. (b) Das Verhältnis von erklärter Varianz ist für die Optdigits-Datensätze aus der UCI-Datensammlung dargestellt. Hierbei handelt es sich um einen Datensatz handschriftlicher Ziffern mit 10 Klassen und 64 dimensionalen Eingaben. Die ersten 20 Eigenvektoren erklären 90% der Varianz.

Die PCA erklärt Varianz und ist empfindlich gegenüber Ausreißern. Einige wenige vom Zentrum entfernte Punkte würden große Auswirkungen auf die Varianzen und somit auf die Eigenvektoren haben. *Robuste Schätzmethoden* erlauben es, Parameter in Gegenwart von Ausreißern zu berechnen. Eine einfache Methode ist die, den Mahalanobis-Abstand der Datenpunkte zu berechnen und diejenigen isolierten Datenpunkte zu verwerfen, die weit weg liegen.

Wenn die ersten zwei Hauptkomponenten einen großen Prozentsatz der Varianz erklären, können wir eine *visuelle Analyse* durchführen. Wir können die Daten im zweidimensionalen Raum abtragen (Abbildung 6.5) und visuell nach Strukturen, Gruppen, Ausreißern, Normalität und so weiter suchen. Diese graphische Darstellung liefert eine bessere bildliche Beschreibung der Stichprobe als die Darstellung für jedes beliebige Paar aus den Originalvariablen. Indem wir die Dimensionen der Hauptkomponenten betrachten, können wir ebenfalls versuchen, bedeutungsvolle zugrundeliegende Variablen, welche die Daten beschreiben, zu entdecken. In Bildbearbeitungsanwendungen beispielsweise, bei denen die Eingaben

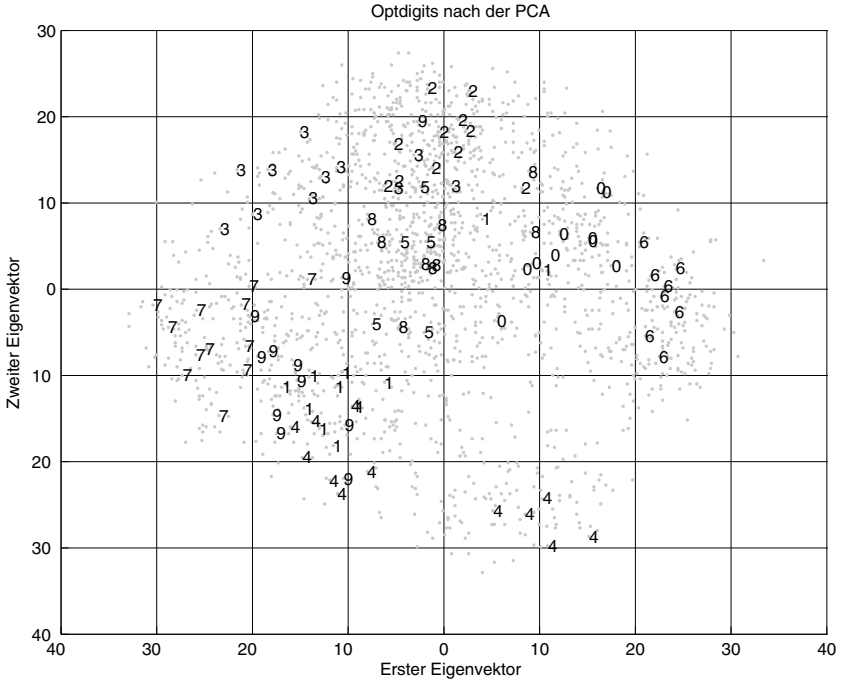


Abb. 6.5: Im Raum von zwei Hauptkomponenten abgetragene Optdigits-Daten. Der Übersicht halber sind nur 100 Datenpunkte mit einem Label versehen.

Bildern entsprechen, können die Eigenvektoren auch als Bilder dargestellt werden und als Vorlagen für wichtige Merkmale gesehen werden; typischerweise spricht man dann von *Eigengesichtern*, *Eigenziffern* und so weiter (Turk und Pentland 1991).

EIGENGESICHT
EIGENZIFFER

Von Gleichung 5.15 wissen wir, dass bei $\boldsymbol{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ nach einer Projektion gilt: $\mathbf{W}^T\boldsymbol{x} \sim \mathcal{N}_k(\mathbf{W}^T\boldsymbol{\mu}, \mathbf{W}^T\boldsymbol{\Sigma}\mathbf{W})$. Im Fall einer d-variaten normalverteilten Stichprobe überführt sie diese in eine k-variate normalverteilte Stichprobe. Das erlaubt eine parametrische Diskriminanzanalyse in diesem Eingaberaum von kleinerer Dimension ($k < d$). Weil z_j unkorreliert sind, ist die neue Kovarianzmatrix diagonal. Wenn sie auf Einheitsvarianz hin normalisiert sind, dann kann die Euklidische Distanz in diesem neuen Raum angewandt werden, was uns zu einem einfacheren Klassifikator führt.

Instanz $\boldsymbol{x}^t$ wird auf den z-Raum projiziert als

$$\boldsymbol{z}^t = \mathbf{W}^T(\boldsymbol{x}^t - \boldsymbol{\mu}) \,.$$

Wenn $\mathbf{W}$ eine orthogonale Matrix ist, so dass $\mathbf{W}\mathbf{W}^T = \mathbf{I}$, dann kann eine Rückprojektion in den Originalraum erfolgen mit

$$\hat{\boldsymbol{x}}^t = \mathbf{W}\boldsymbol{z}^t + \boldsymbol{\mu} \,.$$

$\hat{\boldsymbol{x}}^t$ ist die Rekonstruktion von $\boldsymbol{x}^t$ mittels seiner Repräsentation im z-Raum. Es ist bekannt, dass unter allen orthogonalen linearen Projektionen die PCA den *Rekonstruktionsfehler* minimiert; dieser ergibt sich aus der Distanz zwischen einer Instanz und ihrer Rekonstruktion aus dem niedriger dimensionierten Raum:

Rekonstruktionsfehler

$$\sum_t \|\hat{\boldsymbol{x}}^t - \boldsymbol{x}\|^2 . \tag{6.12}$$

Wie wir bereits weiter vorn erörtert haben, ist der Beitrag eines jeden Eigenvektors durch seinen Eigenwert gegeben. Daher ist es sinnvoll, die Eigenvektoren mit den höchsten Eigenwerten zu behalten. Wenn wir bei der Dimensionalitätsreduktion einige Eigenvektoren mit von null verschiedenen Eigenwerten verwerfen, führt dies zu einem Rekonstruktionsfehler, dessen Größe von den verworfenen Eigenwerten abhängt. Bei Anwendungen zur visuellen Erkennung – beispielsweise der Gesichtserkennung – erlaubt die Darstellung von $\hat{\boldsymbol{x}}^t$ eine visuelle Überprüfung auf Informationsverlust während der PCA.

Die PCA ist nicht überwacht und nutzt keine Ausgabeinformationen. Sie unterscheidet dabei nicht zwischen Eingaben aus unterschiedlichen Gruppen. Bei der Klassifikation jedoch entstammen die Eingaben verschiedenen Gruppen/Klassen. Die *Karhunen-Loève-Expansion* erlaubt die Nutzung von Klasseninformationen; statt etwa die Kovarianzmatrix der gesamten Stichprobe einzusetzen, können wir separate Kovarianzmatrizen für jede Klasse berechnen und deren Durchschnitt (gewichtet mit den a-priori-Wahrscheinlichkeiten der entsprechenden Klassen) als Kovariazmatrix für die PCA benutzen, um daraus die Eigenvektoren zu bestimmen.

Karhunen-Loève-Expansion

Im Falle von *gemeinsamen Hauptkomponenten* (Flury 1988) nehmen wir an, dass die Hauptkomponenten die gleichen für jede Klasse sind, wohingegen die Varianz dieser Komponenten je nach Klasse differiert:

gemeinsame Hauptkomponenten

$$\mathbf{S}_i = \mathbf{C}\mathbf{D}_i\mathbf{C}^T .$$

Dies ermöglicht es, Daten in einem Datenpool zu sammeln, und kann als Regularisierungsmethode angesehen werden, deren Komplexität geringer ist als die einer gemeinsamen Kovarianzmatrix für alle Klassen, aber dennoch die Differenzierung von $\mathbf{S}_i$ erlaubt. Ein verwandter Ansatz ist die *flexible Diskriminanzanalyse* (Hastie, Tibshirani und Buja 1994), welche eine lineare Projektion in einen niedriger dimensionierten Raum vornimmt, wo alle Merkmale unkorreliert sind, und dann einen Minimum-Distanz-Klassifikator nutzt.

flexible Diskriminanzanalyse

6.4 Merkmalseinbettung

Erinnern wir uns, dass $\mathbf{X}$ eine $N \times d$-Matrix ist, wobei N die Anzahl der Instanzen ist und d die Eingabedimensionalität. Die Kovarianzmatrix von

$\boldsymbol{x}$ hat die Dimension $d \times d$, und sie ist gleich $\mathbf{X}^T\mathbf{X}/N$, wenn $\mathbf{X}$ zentriert ist und daher den Mittelwert null hat (ohne Beschränkung der Allgemeinheit). Die Hauptkomponentenanalyse verwendet die Eigenvektoren von $\mathbf{X}^T\mathbf{X}$. Erinnern wir uns, dass die Spektralzerlegung gegeben ist durch

$$\mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{D}\mathbf{W}^T , \tag{6.13}$$

wobei $\mathbf{W}$ eine $d \times d$-Matrix ist, in deren Spalten die Eigenvektoren von $\mathbf{X}^T\mathbf{X}$ stehen, und $\mathbf{D}$ ist eine $d \times d$-Matrix mit den zugehörigen Eigenwerten. Wir nehmen an, dass die Eigenvektoren nach ihren Eigenwerten sortiert sind, so dass in der ersten Spalte von $\mathbf{W}$ der Eigenvektor steht, dessen größter Eigenwert D_{11} ist usw. Wenn $\mathbf{X}^T\mathbf{X}$ den Rang $k < d$ hat, dann ist $D_{ii} = 0$ für $i > k$.

Nehmen wir an, wir wollen die Dimensionalität auf $k < d$ reduzieren. In PCA nehmen wir wie gesagt die ersten k Spalten von $\mathbf{W}$ (diejenigen mit den größten Eigenwerten). Bezeichnen wir diese mit $\boldsymbol{w}_i$ und ihre Eigenwerte mit λ_i, $i = 1, \ldots, k$. Wir bilden auf den neuen k-dimensionalen Raum ab, indem wir das Skalarprodukt der ursprünglichen Eingaben mit den Eigenvektoren bilden:

$$z_i^t = \boldsymbol{w}_i^T\boldsymbol{x}^t, \quad i = 1, \ldots, k, \; t = 1, \ldots, N . \tag{6.14}$$

Sind die λ_i die Eigenwerte und die $\boldsymbol{w}_i$ die Eigenvektoren von $\mathbf{X}^T\mathbf{X}$, dann gilt für jedes $i \leq k$

$$(\mathbf{X}^T\mathbf{X})\boldsymbol{w}_i = \lambda_i\boldsymbol{w}_i .$$

Wenn wir dies von links mit $\mathbf{X}$ multiplizieren, erhalten wir

$$(\mathbf{X}\mathbf{X}^T)\mathbf{X}\boldsymbol{w}_i = \lambda_i\mathbf{X}\boldsymbol{w}_i .$$

Folglich müssen die $\mathbf{X}\boldsymbol{w}_i$ die Eigenvektoren von $\mathbf{X}\mathbf{X}^T$ mit den gleichen Eigenwerten sein (Chatfield und Collins 1980). Man beachte, dass $\mathbf{X}^T\mathbf{X}$ eine $d \times d$-Matrix ist, $\mathbf{X}\mathbf{X}^T$ dagegen eine $N \times N$-Matrix.

Schreiben wir nun ihre Spektralzerlegung auf:

$$\mathbf{X}\mathbf{X}^T = \mathbf{V}\mathbf{E}\mathbf{V}^T . \tag{6.15}$$

$\mathbf{V}$ ist die $N \times N$-Matrix, in deren Spalten die Eigenvektoren von $\mathbf{X}\mathbf{X}^T$ stehen, und $\mathbf{E}$ ist die $N \times N$-Matrix mit den zugehörigen Eigenwerten. Die N-dimensionalen Eigenvektoren von $\mathbf{X}\mathbf{X}^T$ sind die Koordinaten des neuen Raums. Wir bezeichnen dies als *Merkmalseinbettung*.

MERKMALS-EINBETTUNG

An dieser Stelle ein wichtiger Hinweis: Eigenvektoren sind gewöhnlich auf Einheitslänge normiert, so dass für die Eigenvektoren (zu den gleichen Eigenwerten) $\boldsymbol{v}_i$ von $\mathbf{X}\mathbf{X}^T$ gilt

$$\boldsymbol{v}_i = \mathbf{X}\boldsymbol{w}_i/\lambda_i, \quad i = 1, \ldots, k ,$$

da die Summe der Quadrate von $\mathbf{X}\boldsymbol{w}_i$ gleich λ_i ist. Wenn wir also $\boldsymbol{v}_i$ berechnet haben und $\mathbf{X}\boldsymbol{w}_i$ bestimmen wollen (das ist es, was PCA macht), dann müssen wir mit der Quadratwurzel des Eigenwerts multiplizieren:

$$z_i^t = \mathbf{V}_{ti}\sqrt{\mathbf{E}_{tt}}, \quad t = 1, \ldots, N,\ i = 1, \ldots, k\,. \tag{6.16}$$

Wenn $d < N$ gilt, was meist der Fall ist, ist es einfacher, mit $\mathbf{X}^T\mathbf{X}$ zu arbeiten, d. h. PCA zu verwenden. Manchmal gilt $d > N$, und dann ist es einfacher, mit $\mathbf{X}\mathbf{X}^T$ zu arbeiten, was eine $N \times N$-Matrix ist. Beispielsweise haben Gesichtsbilder bei dem als Eigengesichter (Turk und Pentland 1991) bekannten Verfahren der Gesichtserkennung $256 \times 256 = 65\,536$ Pixel, und es gibt nur vierzig Gesichtsbilder (je vier Bilder von 10 Personen). Man beachte, dass der Rang den Wert $\min(d, N)$ niemals überschreiten kann, d. h., bei diesem Verfahren der Gesichtserkennung wissen wir, obwohl die Kovarianzmatrix die Größe $65\,536 \times 65\,536$ hat, dass der Rang (die Anzahl der Eigenvektoren mit Eigenwerten größer 0) niemals den Wert 40 überschreiten kann. Folglich können wir stattdessen mit der 40×40-Matrix arbeiten und die neuen Koordinaten in diesem vierzigdimensionalen Raum verwenden, um zum Beispiel eine Erkennung mithilfe des Nächster-Mittelwert-Klassifikators durchzuführen (Turk und Pentland 1991). Entsprechendes gilt für die meisten Anwendungen der Bioinformatik, wo wir beispielsweise eine lange Gensequenz haben können, aber nur eine kleine Stichprobe. Bei der Clusteranalyse für Texte kann die Anzahl der möglichen Wörter viel größer sein als die Anzahl der Dokumente, und bei einem Empfehlungssysteme für Filme kann die Anzahl der Filme viel größer sein als die der Kunden.

Es gibt allerdings einen Einwand: Im Falle von PCA lernen wir Projektionsvektoren, und wir können jeden neuen Test $\boldsymbol{x}$ in den neuen Raum abbilden, indem wir Skalarprodukte mit den Eigenvektoren bilden – wir haben ein Modell für die Projektion. Mit der Merkmalseinbettung können wir dies nicht tun, da wir keine Projektionsvektoren haben – wir lernen kein Projektionsmodell, sondern erhalten die Koordinaten direkt. Wenn wir neue Testdaten haben, sollten wir sie zu $\mathbf{X}$ hinzufügen und die Berechnung wiederholen.

Das Element (i, j) von $\mathbf{X}\mathbf{X}^T$ ist gleich dem Skalarprodukt der Instanzen i und j, d. h. gleich $(\boldsymbol{x}^i)^T(\boldsymbol{x}^j)$ mit $i, j = 1, \ldots, N$. Wenn wir das Skalarprodukt als Ähnlichkeitsmaß für Vektoren auffassen, dann entspricht $\mathbf{X}\mathbf{X}^T$ einer $N \times N$-Matrix von paarweisen Ähnlichkeiten. In dieser Sichtweise ist die Merkmalseinbettung ein Verfahren, welches Instanzen in einem k-dimensionalen Raum so platziert, dass die paarweisen Ähnlichkeiten in dem neuen Raum die ursprünglichen Ähnlichkeiten widerspiegeln. Wir werden auf diese Idee noch mehrmals zurückkommen: In Abschnitt 6.7 behandeln wir die multidimensionale Skalierung, bei der wir anstelle des Skalarprodukts den Euklidischen Abstand zwischen Vektoren verwenden, und in den Abschnitten 6.10 und 6.12 diskutieren wir mit Isomap

und Laplaceschen Eigenmaps Verfahren, bei denen nicht-Euklidische (Un-)Ähnlichkeitsmaße verwendet werden.

6.5 Faktorenanalyse

Bei der PCA nutzen wir die Originaldimensionen $x_i, i = 1, \ldots, d$ zur Bildung einer neuen Menge an Variablen $\boldsymbol{z}$, welche lineare Kombinationen von x_i sind:

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{\mu}) \,.$$

FAKTORENANALYSE LATENTER FAKTOR

Bei der *Faktorenanalyse* (FA) nehmen wir an, dass es eine Menge an nicht beobachtbaren *latenten Faktoren* $z_j, j = 1, \ldots, k$ gibt, welche bei kombiniertem Auftreten $\boldsymbol{x}$ *generieren*. Somit ist die Richtung entgegengesetzt zur PCA (siehe Abbildung 6.6). Das Ziel ist die Abhängigkeit zwischen den beobachteten Variablen mit Hilfe einer geringeren Zahl an Faktoren zu charakterisieren.

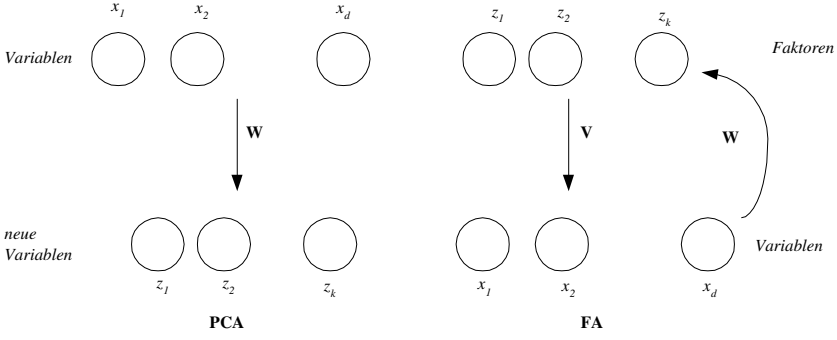


***Abb. 6.6*:** Die Hauptkomponentenanalyse generiert neue Variablen, welche lineare Kombinationen der originalen Eingabevariablen sind. Bei der Faktorenanalyse unterstellen wir jedoch, dass es Faktoren gibt, welche bei linearer Kombination die Eingabevariablen generieren.

Nehmen wir an, es gibt eine Gruppe an Variablen, die untereinander stark korrelieren, aber nur geringe Korrelationen mit anderen Variablen aufweisen. Dann kann es einen einzigen zugrundeliegenden Faktor geben, der zu diesen Variablen führte. Können die anderen Variablen ebenso gruppiert werden, dann können einige wenige Faktoren all diese Variablengruppen repräsentieren. Zwar partitioniert die Faktorenanalyse die Variablen immer in Faktoren-Cluster, ob diese Faktoren jedoch irgendeine Bedeutung haben oder überhaupt existieren, steht dabei in Frage.

Die FA ist genauso wie die PCA eine nicht überwachte Prozedur, die nicht zwischen möglichen Gruppen/Klassen, denen die Datenpunkte

entstammen können, unterscheidet. Das Ziel besteht darin, die Daten in einem Raum mit geringeren Dimensionen ohne Informationsverlust zu modellieren. Bei der FA wird dies als Korrelation zwischen Variablen gemessen.

Wie bei der PCA haben wir eine Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_t$, die einer unbekannten Wahrscheinlichkeitsdichte mit $E[\boldsymbol{x}] = \boldsymbol{\mu}$ und $\text{Cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$ entnommen folgt *oder* gehorcht. Wir nehmen an, dass die Faktoren jeweils einer Einheitsnormalverteilung gehorchen mit $E[z_j] = 0, \text{Var}(z_j) = 1$, und dass keine Korrelationen vorliegen, also $\text{Cov}(z_i, z_j) = 0, i \neq j$. Um für das Rechnung zu tragen, was nicht durch die Faktoren erklärt wird, gibt es eine hinzugefügte Quelle für jede Eingabe, welche wir mit ϵ_i bezeichnen. Es wird davon ausgegangen, dass sie einen Nullmittelwert, $E[\epsilon_i] = 0$, sowie eine unbekannte Varianz, $\text{Var}(\epsilon_i) = \psi_i$, hat. Diese spezifischen Quellen sind untereinander nicht korreliert, $\text{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$; auch Korrelationen mit den Faktoren liegen nicht vor: $\text{Cov}(\epsilon_i, z_j) = 0, \forall i, j$.

Die FA nimmt an, dass jede Eingabedimension, $x_i, i = 1, \ldots, d$, als die gewichtete Summe der $k < d$ Faktoren, $z_j, j = 1, \ldots, k$, plus den Restterm geschrieben werden kann (siehe Abbildung 6.7):

$$x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \cdots + v_{ik}z_k + \epsilon_i, \forall i = 1, \ldots, d\,,$$
$$x_i - \mu_i = \sum_{j=1}^{k} v_{ij}z_j + \epsilon_i\,. \tag{6.17}$$

Dies kann in der Vektormatrixform ausgedrückt werden als

$$\boldsymbol{x} - \boldsymbol{\mu} = \mathbf{V}\boldsymbol{z} + \boldsymbol{\epsilon}\,, \tag{6.18}$$

wobei $\mathbf{V}$ die $d \times k$ Matrix der Gewichte, auch *Faktorenladungen* genannt, ist. Von nun an werden wir ohne Beschränkung der Allgemeinheit $\boldsymbol{\mu} = \mathbf{0}$ annehmen; wir können $\boldsymbol{\mu}$ jederzeit nach der Projektion hinzufügen. Unter Beachtung, dass $\text{Var}(z_j) = 1$ und $\text{Var}(\epsilon_i) = \psi_i$, gilt:

$$\text{Var}(x_i) = v_{i1}^2 + v_{i2}^2 + \cdots + v_{ik}^2 + \psi_i\,. \tag{6.19}$$

$\sum_{j=1}^{k} v_{ij}^2$ ist der Teil der Varianz, welcher durch die gemeinsamen Faktoren erklärt wird, und ψ_i ist die für x_i spezifische Varianz.

In der Vektormatrixform ausgedrückt erhalten wir

$$\begin{aligned}\boldsymbol{\Sigma} = \text{Cov}(\boldsymbol{x}) &= \text{Cov}(\mathbf{V}\boldsymbol{z} + \boldsymbol{\epsilon}) &(6.20)\\ &= \text{Cov}(\mathbf{V}\boldsymbol{z}) + \text{Cov}(\boldsymbol{\epsilon})\\ &= \mathbf{V}\text{Cov}(\boldsymbol{z})\mathbf{V}^T + \boldsymbol{\Psi}\\ &= \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}\,, &(6.21)\end{aligned}$$

wobei $\boldsymbol{\Psi}$ eine diagonale Matrix mit ψ_i an den Diagonalen ist. Da die Faktoren unkorreliert sind und jeweils einer Standardnormalverteilung

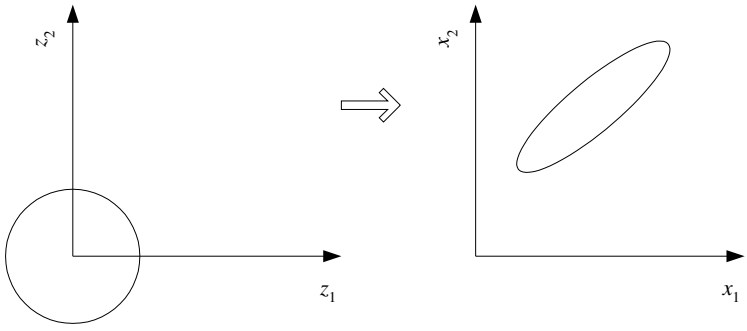


***Abb. 6.7*:** Die Faktoren sind unabhängig und gehorchen jeweils einer Standardnormalverteilung. Sie werden gestreckt, rotiert und verschoben, um die Eingabe zu erhalten.

gehorchen, erhalten wir $\text{Cov}(\boldsymbol{z}) = \mathbf{I}$. Bei zwei Faktoren zum Beispiel ist

$$\text{Cov}(x_1, x_2) = v_{11}v_{21} + v_{12}v_{22} \,.$$

Wenn x_1 und x_2 eine hohe Kovarianz aufweisen, stehen sie über einen gemeinsamen Faktor in Beziehung. Wenn dies der erste Faktor ist, so sind sowohl v_{11} als auch v_{21} hoch; wenn es der zweite Faktor ist, so sind v_{12} und v_{22} beide hoch. In beiden Fällen ist die Summe $v_{11}v_{21} + v_{12}v_{22}$ hoch. Wenn die Kovarianz niedrig ausfällt, dann hängen x_1 und x_2 von unterschiedlichen Faktoren ab, und in den Produkten der Summe wird ein Term hoch ausfallen, der andere niedrig; auch die Summe wird gering ausfallen.

Wir erkennen, dass

$$\text{Cov}(x_1, z_2) = \text{Cov}(v_{12}z_2, z_2) = v_{12}\text{Var}(z_2) = v_{12} \,.$$

Somit ist $\text{Cov}(\boldsymbol{x}, \boldsymbol{z}) = \mathbf{V}$ und es wird deutlich, dass die Ladungen die Korrelationen zwischen den Variablen und den Faktoren repräsentieren.

Bei gegebenem $\mathbf{S}$, dem Schätzer von $\boldsymbol{\Sigma}$, wollen wir nun gerne $\mathbf{V}$ und $\boldsymbol{\Psi}$ finden, so dass

$$\mathbf{S} = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi} \,.$$

Wenn es nur wenige Faktoren gibt, sprich wenn $\mathbf{V}$ eine geringe Spaltenanzahl aufweist, so haben wir es mit einer vereinfachten Struktur für $\mathbf{S}$ zu tun, da $\mathbf{V}$ $d \times k$ ist und $\boldsymbol{\Psi}$ genau d Werte besitzt, wodurch somit die Anzahl an Parametern von d^2 auf $d \cdot k + d$ reduziert wird.

Da $\boldsymbol{\Psi}$ diagonal ist, werden Kovarianzen durch $\mathbf{V}$ repräsentiert. Man beachte, dass die PCA kein separates $\boldsymbol{\Psi}$ gestattet und versucht, sowohl

die Kovarianzen *als auch* die Varianzen in die Betrachtung einzubeziehen. Wenn alle ψ_i gleich sind, also $\mathbf{\Psi} = \psi\mathbf{I}$, ergibt sich die *probabilistische PCA* (Tipping und Bishop 1999); falls gilt ψ_i gleich 0, ergibt sich die konventionelle PCA.

PROBABILISTISCHE PCA

Befassen wir uns nun damit, wie wir die Faktorenladungen und die spezifischen Varianzen herausfinden können. Dazu ignorieren wir $\mathbf{\Psi}$ zunächst einmal. Anhand seiner Spektralzerlegung wissen wir dann, dass

$$\mathbf{S} = \mathbf{CDC}^T = \mathbf{CD}^{1/2}\mathbf{D}^{1/2}\mathbf{C} = (\mathbf{CD}^{1/2})(\mathbf{CD}^{1/2})^T ,$$

wobei wir nur k der Eigenvektoren hernehmen, indem wir das Verhältnis der erklärten Varianz betrachten, so dass $\mathbf{C}$ die $d \times k$ Matrix aus Eigenvektoren ist und $\mathbf{D}^{1/2}$ der $k \times k$ Diagonalenmatrix mit den Quadratwurzeln der Eigenwerte an ihren Diagonalen entspricht. Somit erhalten wir

$$\mathbf{V} = \mathbf{CD}^{1/2} . \tag{6.22}$$

Wir können ψ_j anhand von Gleichung 6.19 herausfinden als

$$\psi_i = s_i^2 - \sum_{j=1}^{k} v_{ij}^2 . \tag{6.23}$$

Man beachte, dass die Multiplikation von $\mathbf{V}$ mit einer beliebigen orthogonalen Matrix, und zwar mit der Eigenschaft $\mathbf{TT}^T = \mathbf{I}$, eine weitere zulässige Lösung darstellt und die Lösung somit nicht eindeutig ist.

$$\mathbf{S} = (\mathbf{VT})(\mathbf{VT})^T = \mathbf{VTT}^T\mathbf{V}^T = \mathbf{VIV}^T = \mathbf{VV}^T$$

Wenn $\mathbf{T}$ eine orthogonale Matrix ist, ändert sich die Distanz zum Ursprung nicht. Wenn $\boldsymbol{z} = \mathbf{T}\boldsymbol{x}$, dann gilt:

$$\boldsymbol{z}^T\boldsymbol{z} = (\mathbf{T}\boldsymbol{x})^T(\mathbf{T}\boldsymbol{x}) = \boldsymbol{x}^T\mathbf{T}^T\mathbf{T}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} .$$

Die Multiplikation mit einer orthogonalen Matrix hat den Effekt, die Achsen zu rotieren, wodurch uns ermöglicht wird, die Menge an Achsen zu wählen, welche am besten zu interpretieren sind (Rencher 1995). In zwei Dimensionen rotiert

$$\mathbf{T} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

die Achsen um ϕ. Es gibt zwei Arten von Rotation. Bei der orthogonalen Rotation sind die Faktoren auch nach der Rotation immer noch orthogonal; bei der schiefen Rotation hingegen können die Faktoren Korrelationen entwickeln. Die Faktoren werden rotiert, um die maximale Ladung auf so wenige Faktoren pro Variable wie möglich zu verteilen, um die Faktoren

interpretierbar zu gestalten. Allerdings ist die Interpretierbarkeit subjektiv und sollte nicht dazu genutzt werden, die eigenen Vorurteile in die Daten hinzuzwingen.

Es gibt zwei mögliche Anwendungsgebiete der Faktorenanalyse. Zum einen kann sie für die Wissensextraktion genutzt werden, wenn wir die Ladungen herausfinden und versuchen, die Variablen mit weniger Faktoren auszudrücken. Sie kann aber auch für die Dimensionalitätsreduktion eingesetzt werden, wenn $k < d$. Wir haben bereits gesehen, wie ersteres vollzogen wird. Betrachten wir daher nun, wie die Faktorenanalyse bei der Dimensionalitätsreduktion eingesetzt werden kann.

Wenn wir die Verringerung der Dimensionalität zum Ziel haben, müssen wir in der Lage sein, die Scores der Faktoren, z_j, aus x_i zu finden. Wir wollen die Ladungen w_{ji} herausfinden, bei denen gilt:

$$z_j = \sum_{i=1}^{d} w_{ji} x_i + \epsilon_i, j = 1, \ldots, k \,, \tag{6.24}$$

wobei x_i zentriert sind, so dass sie einen Nullmittelwert aufweisen. In Vektorform kann dies für die Beobachtung t notiert werden als

$$\boldsymbol{z}^t = \mathbf{W}^T \boldsymbol{x}^t + \boldsymbol{\epsilon}, \forall t = 1, \ldots, N \,.$$

Hierbei handelt es sich um ein lineares Modell mit d Eingaben und k Ausgaben. Seine Transponierte kann wie folgt geschrieben werden:

$$(\boldsymbol{z}^t)^T = (\boldsymbol{x}^t)^T \mathbf{W} + \boldsymbol{\epsilon}^T, \forall t = 1, \ldots, N \,.$$

Vorausgesetzt, wir haben eine Stichprobe mit N Beobachtungen, schreiben wir

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \boldsymbol{\Xi} \,, \tag{6.25}$$

wobei $\mathbf{Z}$ aus $N \times k$ Faktoren, $\mathbf{X}$ aus $N \times d$ (zentrierten) Beobachtungen und $\boldsymbol{\Xi}$ aus $N \times k$ Nullmittelwertrauschen zusammengesetzt sind. Dies ist eine multivariate lineare Regression mit multiplen Ausgaben, und basierend auf Abschnitt 5.8 wissen wir, dass $\mathbf{W}$ durch

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Z}$$

gefunden werden kann, wir aber $\mathbf{Z}$ nicht kennen; genau das möchten wir gerne berechnen. Wir multiplizieren und dividieren beide Seiten mit $N - 1$ und erhalten

$$\begin{aligned} \mathbf{W} &= (N-1)(\mathbf{X}^T\mathbf{X})^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \left(\frac{\mathbf{X}^T\mathbf{X}}{N-1}\right)^{-1}\frac{\mathbf{X}^T\mathbf{Z}}{N-1} \\ &= \mathbf{S}^{-1}\mathbf{V} \,. \end{aligned} \tag{6.26}$$

Setzen wir Gleichung 6.26 in Gleichung 6.25 ein, schreiben wir

$$\mathbf{Z} = \mathbf{XW} = \mathbf{XS}^{-1}\mathbf{V} , \tag{6.27}$$

davon ausgehend, dass $\mathbf{S}$ nicht singulär ist. Man kann $\mathbf{R}$ anstelle von $\mathbf{S}$ benutzen, wenn x_i normalisiert sind, so dass sie der Einheitsvarianz unterliegen.

Für die Dimensionalitätsreduktion bietet die FA keinen Vorteil gegenüber der PCA, außer der Interpretierbarkeit der Faktoren, welche die Identifikation von gemeinsamen Ursachen, eine einfache Erklärung und Wissensextraktion ermöglicht. Im Kontext der Spracherkennung beispielsweise korrespondiert $\boldsymbol{x}$ mit dem akustischen Signal. Wir wissen jedoch, dass dies das Ergebnis der (nichtlinearen) Interaktion einer geringen Anzahl von *Artikulatoren* ist, nämlich Kiefer, Zunge, Gaumen, Lippen und Mund, welche so positioniert sind, dass sie die Luft beim Austreten aus der Lunge formen können, um den Sprachklang zu generieren. Wenn ein Sprachsignal in diesen artikulatorischen Raum transformiert werden könnte, dann wäre die Erkennung deutlich einfacher. Die Verwendung solcher generativen Modelle ist eine der gegenwärtigen Forschungsrichtungen in der Spracherkennung. In Kapitel 14 diskutieren wir, wie solche Modelle grafisch repräsentiert werden können.

6.6 Singulärwertzerlegung und Faktorisierung von Matrizen

Wenn wir eine $N \times d$-Matrix $\mathbf{X}$ gegeben haben, dann arbeiten wir mit $\mathbf{X}^T\mathbf{X}$, falls $d < N$, oder wir arbeiten mit $\mathbf{XX}^T$, falls $N < d$. Bei beiden handelt es sich um quadratische Matrizen, und die Spektralzerlegung liefert uns $\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$. Hierbei ist $\mathbf{Q}$, die Matrix der Eigenvektoren, orthogonal ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$) und in der Diagonale von $\boldsymbol{\Lambda}$ stehen die Eigenwerte.

Die *Singulärwertzerlegung* gestattet es uns, beliebige (auch nichtquadratische) $N \times d$-Matrizen zu zerlegen (Strang 2006): SINGULÄRWERTZERLEGUNG

$$\mathbf{X} = \mathbf{VAW}^T . \tag{6.28}$$

Dabei ist $\mathbf{V}$ eine $N \times N$-Matrix, in deren Spalten die Eigenvektoren von $\mathbf{XX}^T$ stehen, und $\mathbf{W}$ ist eine $d \times d$-Matrix, in deren Spalten die Eigenvektoren von $\mathbf{X}^T\mathbf{X}$ stehen. Die $N \times d$-Matrix $\mathbf{A}$ enthält in ihrer Diagonale die $k = \min(N, d)$ *Singulärwerte* $a_i, i = 1, \ldots, k$, welche die Quadratwurzeln der von null verschiedenen Eigenwerte sowohl von $\mathbf{XX}^T$ als auch von $\mathbf{X}^T\mathbf{X}$ sind; die übrigen Elemente der Matrix $\mathbf{A}$ sind null. $\mathbf{V}$ und $\mathbf{W}^T$ sind orthogonale Matrizen (aber nicht notwendig Transponierte voneinander).

$$\mathbf{XX}^T = (\mathbf{VAW}^T)(\mathbf{VAW}^T)^T = \mathbf{VAW}^T\mathbf{WA}^T\mathbf{V}^T = \mathbf{VEV}^T$$
$$\mathbf{X}^T\mathbf{X} = (\mathbf{VAW}^T)^T(\mathbf{VAW}^T) = \mathbf{WA}^T\mathbf{V}^T\mathbf{VAW}^T = \mathbf{WDW}^T .$$

Hierbei ist $\mathbf{E} = \mathbf{A}\mathbf{A}^T$ und $\mathbf{D} = \mathbf{A}^T\mathbf{A}$. Diese Matrizen haben unterschiedliche Größen, doch beide sind sie quadratisch, wobei in ihren Diagonalen die $a_i, i = 1, \ldots, k$ stehen und alle anderen Elemente null sind.

Genauso wie in Gleichung 6.10 können wir schreiben

$$\mathbf{X} = \boldsymbol{u}_1 a_1 \boldsymbol{v}_1^T + \boldsymbol{u}_2 a_2 \boldsymbol{v}_2^T + \cdots \boldsymbol{u}_k a_k \boldsymbol{v}_k^T \,. \tag{6.29}$$

Wir können die zu sehr kleinen, aber von null verschiedenen a_i gehörenden $\boldsymbol{u}_i$ und $\boldsymbol{v}_i$ ignorieren und dennoch ohne allzu großen Fehler $\mathbf{X}$ rekonstruieren.

MATRIX-FAKTORISIERUNG

Bei der *Matrixfaktorisierung* schreiben wir eine sehr große Matrix als Produkt zweier Matrizen:

$$\mathbf{X} = \mathbf{F}\mathbf{G} \,. \tag{6.30}$$

Hierbei ist $\mathbf{X}$ eine $N \times d$-Matrix, $\mathbf{F}$ eine $N \times k$-Natrix und $\mathbf{G}$ eine $k \times d$-Matrix. k ist die Dimensionalität des Faktorraums und hoffentlich viel kleiner als d und N. Die Idee ist die, dass, obwohl die Datenmenge möglicherweise sehr groß ist, die Matrix dünn besetzt ist bzw. dass es starke Korrelationen gibt und daher eine Darstellung in einem Raum von geringerer Dimension möglich ist.

$\mathbf{G}$ definiert Faktoren auf der Basis der ursprünglichen Attribute und $\mathbf{F}$ definiert Dateninstanzen auf der Basis dieser Faktoren. Ist $\mathbf{X}$ beispielsweise eine Stichprobe aus N Dokumenten, von denen jedes eine Bag-of-Words-Darstellung mit d Wörtern hat, dann kann jeder der Faktoren ein Thema oder ein Konzept sein, das mit einer bestimmten Teilmenge der Wörter beschrieben wird, und jedes Dokument ist eine bestimmte Kombination solcher Faktoren. Dies wird als *latent semantische Indexierung* bezeichnet (Landauer, Laham und Derr 2004). Bei der nichtnegativen Matrixfaktorisierung sind die Matrizen nichtnegativ, was die Repräsentation eines komplexen Objektes mithilfe seiner Teile gestattet (Lee und Seung 1999).

LATENT SEMANTISCHE INDEXIERUNG

Betrachten wir ein anderes Beispiel aus dem Einzelhandel, wobei $\mathbf{X}$ wieder die Kundendaten sind. Wir haben N Kunden und wir verkaufen d verschiedene Produkte. $\mathbf{X}_{ti}$ entspricht der Menge des Produkts i, die der Kunde t gekauft hat. Wir wissen, dass Kunden Produkte nicht zufällig kaufen und dass ihre Einkäufe von einer Reihe von Faktoren abhängen, etwa von der Größe und der Zusammensetzung ihres Haushalts, vom Einkommen, von ihren Vorlieben usw. – diese Faktoren sind uns im Allgemeinen verborgen. Bei der Matrixfaktorisierung von Kundendaten nehmen wir an, dass, dass es k solche Faktoren gibt. $\mathbf{G}$ verbindet Faktoren mit Produkten: $\mathbf{G}_j$ ist ein d-dimensionaler Vektor, der die Beziehung zwischen Faktor j und den einzelnen Produkten beschreibt. Genauer gesagt ist $\mathbf{G}_{ji}$ proportional zur Menge des Produkts i, die aufgrund des Faktors j gekauft wurde. Entsprechend verbindet $\mathbf{F}$ Kunden mit Faktoren: $\mathbf{F}_t$ ist der k-dimensionale Vektor, der den Kunden t durch

die verborgenen Faktoren beschreibt. Genauer gesagt drückt $\mathbf{F}_{tj}$ unsere Vermutung darüber aus, wie sehr das Verhalten des Kunden t durch den Faktor j bestimmt ist. Wir können dann Gleichung 6.30 umformen zu

$$\mathbf{X}_{ti} = \mathbf{F}_t^T \mathbf{G}_i = \sum_{j=1}^{k} \mathbf{F}_{tj} \mathbf{G}_{ji} \,. \tag{6.31}$$

Das heißt, um die Gesamtmenge zu berechnen, bilden wir die Sume über alle Faktoren, wobei wir für jeden einzelnen unsere Annahme darüber, wie sehr der Kunde von diesem Faktor beeinflusst ist, mit der aufgrund dieses Faktors gekauften Menge multiplizieren (siehe Abbildung 6.8).

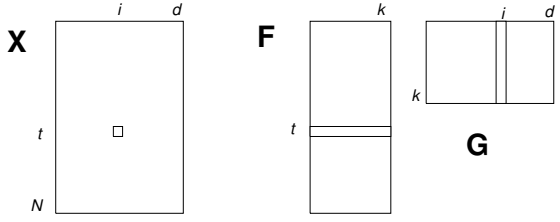


***Abb. 6.8*:** Matrixfaktorisierung. $\mathbf{X}$ ist die $N \times d$-Datenmatrix. $\mathbf{F}$ ist eine $N \times k$-Matrix und ihre Zeile t definiert die Instanz t durch die k verborgenen Faktoren. Die $k \times d$-Matrix $\mathbf{G}$ beschreibt die Faktoren durch die d beobachteten Variablen. Um $\mathbf{X}_{ti}$ zu erhalten, betrachten wir alle k Faktoren und bilden eine gewichtete Summe über sie.

6.7 Multidimensionale Skalierung

Nehmen wir einmal an, dass uns für N Punkte die Distanzen zwischen Punktpaaren vorliegen: $d_{ij}, i, j = 1, \ldots, N$. Wir kennen weder die exakten Koordinaten der Punkte noch ihre Dimensionalität, noch wissen wir, wie die Distanzen berechnet werden. Die *multidimensionale Skalierung* (MDS) ist die Methode, mit der diese Punkte in einem Raum niedriger Dimensionalität – beispielsweise in einem zweidimensionalen – platziert werden, so dass die Euklidische Distanz zwischen ihnen im zweidimensionalen Raum so nah als möglich an d_{ij} liegt, den gegebenen Distanzen im Originalraum. Somit bedarf es einer Projektion von einem Raum unbekannter Dimensionalität in einen Raum von beispielsweise zwei Dimensionen.

MULTIDIMENSIONALE SKALIERUNG

Im archetypischen Beispiel der multidimensionalen Skalierung nehmen wir die Reiseentfernungen zwischen Städten zur Hand und erhalten nach der Anwendung der MDS eine Approximation an eine entsprechende Landkarte. Die Karte ist verfälscht, und zwar dahingehend, dass die Karte an Landesteilen mit geographischen Hindernissen wie Bergen und

Seen, bei denen die Reiseentfernung auf dem Landweg deutlich vom direkten Luftweg abweicht, ausgedehnt wird, um längere Entfernungen zu berücksichtigen (siehe Abbildung 6.9). Die Karte ist um den Ursprung zentriert, jedoch ist die Lösung noch nicht eindeutig. Wir können beliebige rotierte oder gespiegelte Versionen herstellen.

Die MDS kann zur Dimensionalitätsreduktion genutzt werden, indem paarweise die Euklidischen Distanzen im d-dimensionalen $\boldsymbol{x}$-Raum berechnet und als Eingabe an die MDS geleitet werden, welche das Ganze dann in einen niedriger dimensionierten Raum projiziert, wobei die Distanzen erhalten bleiben.

Sagen wir, wir haben eine Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ wie üblich, wobei $\boldsymbol{x}^t \in \mathbb{R}^d$. Für zwei Punkte r und s ergibt sich die quadrierte Euklidische Distanz zwischen ihnen als

$$\begin{aligned} d_{rs}^2 &= \|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2 = \sum_{j=1}^{d} (x_j^r - x_j^s)^2 = \sum_{j=1}^{d} (x_j^r)^2 - 2\sum_{j=1}^{d} x_j^r x_j^s + \sum_{j=1}^{d} (x_j^s)^2 \\ &= b_{rr} + b_{ss} - 2b_{rs} \,, \end{aligned} \tag{6.32}$$

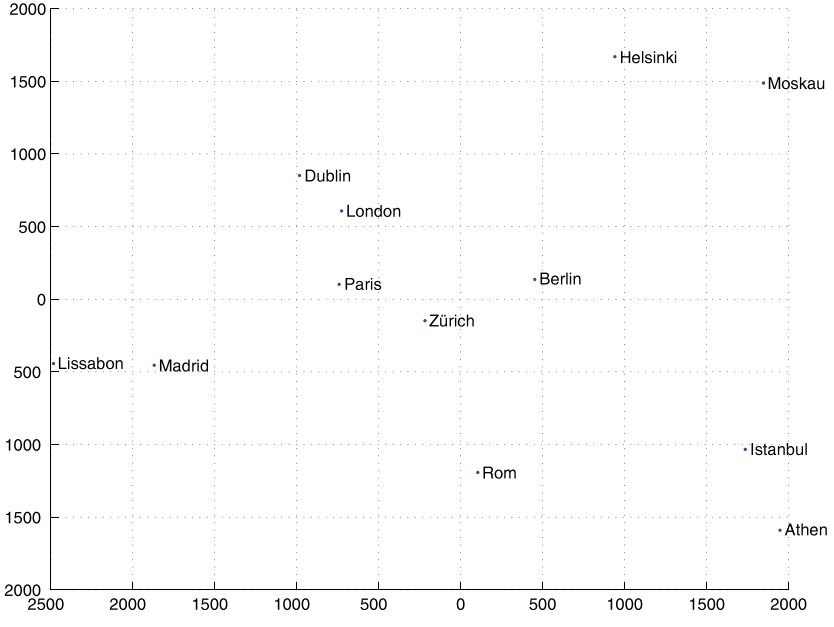


***Abb. 6.9*:** Mit MDS dargestellte Karte von Europa. Paarweise Entfernungen zwischen diesen Städten auf dem Landweg werden als Eingabe genommen und durch die MDS in zwei Dimensionen platziert, und zwar so, dass die Distanzen so gut wie möglich erhalten bleiben.

wobei b_{rs} definiert ist als

$$b_{rs} = \sum_{j=1}^{d} x_j^r x_j^s \,. \tag{6.33}$$

Um die Lösung einzugrenzen, zentrieren wir die Daten am Ursprung und nehmen an, dass

$$\sum_{t=1}^{N} x_j^t = 0, \forall j = 1, \ldots, d \,.$$

Wenn wir dann Gleichung 6.32 an r, s sowie an beiden, r, s, zusammenfassen und definieren, dass

$$T = \sum_{t=1}^{N} b_{tt} = \sum_t \sum_j (x_j^t)^2 \,,$$

dann erhalten wir

$$\begin{aligned} \sum_r d_{rs}^2 &= T + N b_{ss} \,, \\ \sum_s d_{rs}^2 &= N b_{rr} + T \,, \\ \sum_r \sum_s d_{rs}^2 &= 2NT \,. \end{aligned}$$

Wir definieren nun

$$d_{\bullet s}^2 = \frac{1}{N} \sum_r d_{rs}^2 \,, \quad d_{r\bullet}^2 = \frac{1}{N} \sum_s d_{rs}^2 \,, \quad d_{\bullet\bullet}^2 = \frac{1}{N^2} \sum_r \sum_s d_{rs}^2$$

und nutzen dann Gleichung 6.32. Dadurch erhalten wir

$$b_{rs} = \frac{1}{2}(d_{r\bullet}^2 + d_{\bullet s}^2 - d_{\bullet\bullet}^2 - d_{rs}^2) \,. \tag{6.34}$$

Nachdem wir nun b_{rs} berechnet haben und wissen, dass laut Definition in Gleichung 6.33 $\mathbf{B} = \mathbf{X}\mathbf{X}^T$, können wir die Merkmalseinbettung anwenden (siehe Abschnitt 6.4). Wir wissen anhand der Spektralzerlegung, dass $\mathbf{X} = \mathbf{C}\mathbf{D}^{1/2}$ als Approximation für $\mathbf{X}$ genutzt werden kann, wobei $\mathbf{C}$ die Matrix ist, deren Spalten den Eigenvektoren von $\mathbf{B}$ entsprechen und $\mathbf{D}^{1/2}$ eine diagonale Matrix mit den Quadratwurzeln der Eigenwerte an den Diagonalen ist. Indem wir die Eigenwerte von $\mathbf{B}$ betrachten, können wir uns für eine Dimensionalität k entscheiden, die niedriger ist als d (und

N), genauso wie wir das bei der PCA und FA getan haben. Sagen wir, $\boldsymbol{c}_j$ seien die Eigenvektoren mit λ_j als korrespondierenden Eigenwerten. Man beachte, dass $\boldsymbol{c}_j$ N-dimensional ist. Somit erhalten wir die neuen Dimensionen als

$$z_j^t = \sqrt{\lambda_j} c_j^t, j = 1, \ldots, k, t = 1, \ldots, N \,. \tag{6.35}$$

Das heißt, die neuen Koordinaten von Instanz t sind durch die t-ten Elemente der Eigenvektoren $\boldsymbol{c}_j, j = 1, \ldots, k$ nach der Normierung gegeben.

Wir wissen, dass die Hauptkomponentenanalyse und die Merkmalseinbettung das gleiche leisten. Dies zeigt, dass die PCA dieselbe Arbeit wie die MDS durchführt und dies mit geringeren Kosten tut. Eine PCA, angewandt auf die Korrelationsmatrix statt auf die Kovarianzmatrix, entspricht der Anwendung der MDS mit standardisierten Euklidischen Distanzen, wenn jede Variable Einheitsvarianz besitzt.

Im allgemeinen Fall wollen wir eine Abbildung $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x}|\theta)$ finden, wobei $\boldsymbol{z} \in \mathbb{R}^k, \boldsymbol{x} \in \mathbb{R}^d$ und $\boldsymbol{g}(\boldsymbol{x}|\theta)$ die Abbildungsfunktion von d auf k Dimensionen ist, die bis auf die Menge von Parametern θ definiert ist. Die klassische MDS, die wir bereits besprochen haben, entspricht einer linearen Transformation

$$\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T \boldsymbol{x} \,, \tag{6.36}$$

Sammon-Mapping

aber im allgemeinen Fall kann auch eine nichtlineare Abbildung genutzt werden; hierbei spricht man vom *Sammon-Mapping* oder Sammon-Verfahren. Der normalisierte Fehler bei der Abbildung wird als *Sammon-Stress* bezeichnet und definiert sich als

$$\begin{aligned} E(\theta|\mathcal{X}) &= \sum_{r,s} \frac{(\|\boldsymbol{z}^r - \boldsymbol{z}^s\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2} \\ &= \sum_{r,s} \frac{(\|\boldsymbol{g}(\boldsymbol{x}^r|\theta) - \boldsymbol{g}(\boldsymbol{x}^s|\theta)\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|)^2}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2} \,. \end{aligned} \tag{6.37}$$

Man kann eine beliebige Regressionsmethode für $\boldsymbol{g}(\cdot|\theta)$ nutzen und θ schätzen, um den Sammon-Stress für den Trainingsdatensatz $\mathcal{X}$ zu minimieren. Wenn $\boldsymbol{g}(\cdot)$ in $\boldsymbol{x}$ nichtlinear ist, so entspricht dies einer nichtlinearen Dimensionalitätsreduktion.

Im Falle der Klassifikation können Klasseninformationen in die Distanz einbezogen werden (siehe Webb 1999) als

$$d'_{rs} = (1 - \alpha) d_{rs} + \alpha c_{rs} \,,$$

wobei c_{rs} die „Distanz“ zwischen den Klassen ist, zu denen $\boldsymbol{x}^r$ und $\boldsymbol{x}^s$ gehören. Diese Interklassendistanz sollte subjektiv bereitgestellt werden, und α wird unter Zuhilfenahme der Kreuzvalidierung optimiert.

6.8 Lineare Diskriminanzanalyse

Die *lineare Diskriminanzanalyse* (LDA) ist eine überwachte Methode zur Dimensionalitätsreduktion für Klassifikationsprobleme. Wir beginnen mit einem Fall mit zwei Klassen und verallgemeinern dann auf $K > 2$ Klassen. LINEARE DISKRIMINANZANALYSE

Bei gegebenen Stichproben aus zwei Klassen $\mathcal{C}_1$ und $\mathcal{C}_2$ wollen wir die Richtung herausfinden, definiert durch den Vektor $\boldsymbol{w}$, bei welcher eine Projektion der Daten auf $\boldsymbol{w}$ dazu führt, dass die Beispiele aus den zwei Klassen so gut wie möglich voneinander getrennt liegen. Wie wir bereits gesehen haben, ist

$$z = \boldsymbol{w}^T \boldsymbol{x} \tag{6.38}$$

die Projektion von $\boldsymbol{x}$ auf $\boldsymbol{w}$ und somit eine Dimensionalitätsreduktion von d auf 1.

$\boldsymbol{m}_1$ und m_1 sind die Mittelwerte von Stichproben aus $\mathcal{C}_1$ vor beziehungsweise nach der Projektion. Man beachte, dass $\boldsymbol{m}_1 \in \mathbb{R}^d$ und $m_1 \in \mathbb{R}$. Wir erhalten eine Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$, so dass $r^t = 1$, wenn $\boldsymbol{x}^t \in \mathcal{C}_1$ und $r^t = 0$, falls $\boldsymbol{x}^t \in \mathcal{C}_2$.

$$\begin{aligned} m_1 &= \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t r^t}{\sum_t r^t} = \boldsymbol{w}^T \boldsymbol{m}_1 \\ m_2 &= \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t (1 - r^t)}{\sum_t (1 - r^t)} = \boldsymbol{w}^T \boldsymbol{m}_2 \end{aligned} \tag{6.39}$$

Die *Streuung* von Stichproben aus $\mathcal{C}_1$ und $\mathcal{C}_2$ nach der Projektion ist STREUUNG

$$\begin{aligned} s_1^2 &= \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_1)^2 r^t \,, \\ s_2^2 &= \sum_t (\boldsymbol{w}^T \boldsymbol{x}^t - m_2)^2 (1 - r^t) \,. \end{aligned} \tag{6.40}$$

Damit die zwei Klassen bestmöglich getrennt sind, wollen wir, dass die Mittelwerte nach der Projektion so weit auseinander liegen wie möglich und dass die Beispiele der Klassen in einer kleinstmöglichen Region verstreut liegen. Somit erhoffen wir uns einen möglichst großen Wert für $|m_1 - m_2|$ und einen kleinen für $s_1^2 + s_2^2$ (siehe Abbildung 6.10). *Fishers lineare Diskriminante* ist der Wert für $\boldsymbol{w}$, wodurch FISHERS LINEARE DISKRIMINANTE

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{6.41}$$

maximiert wird.

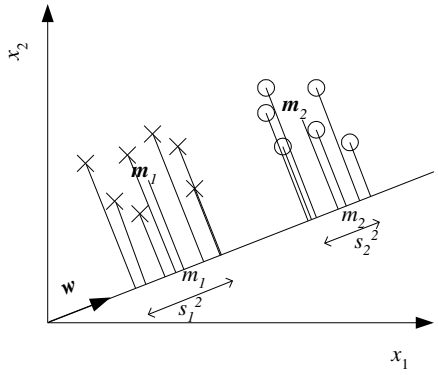


Abb. 6.10: Zweidimensionale Zweiklassendaten projiziert auf $\boldsymbol{w}$.

Wenn wir den Zähler umformen, erhalten wir

$$\begin{aligned}(m_1 - m_2)^2 &= (\boldsymbol{w}^T\boldsymbol{m}_1 - \boldsymbol{w}^T\boldsymbol{m}_2)^2 \\ &= \boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T\boldsymbol{w} \\ &= \boldsymbol{w}^T\mathbf{S}_B\boldsymbol{w}\,, \end{aligned} \tag{6.42}$$

INTERKLASSENSTREUUNGSMATRIX

wobei $\mathbf{S}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T$ die *Matrix der Interklassenstreuung* ist. Der Nenner ist die Summe der Streuungen von Beispielen von Klassen um ihre Mittelwerte nach der Projektion; er kann umgeformt werden in

$$\begin{aligned}s_1^2 &= \sum_t (\boldsymbol{w}^T\boldsymbol{x}^t - m_1)^2 r^t \\ &= \sum_t \boldsymbol{w}^T(\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T\boldsymbol{w}r^t \\ &= \boldsymbol{w}^T\mathbf{S}_1\boldsymbol{w}\,, \end{aligned} \tag{6.43}$$

wobei

$$\mathbf{S}_1 = \sum_t r^t(\boldsymbol{x}^t - \boldsymbol{m}_1)(\boldsymbol{x}^t - \boldsymbol{m}_1)^T \tag{6.44}$$

INTRAKLASSENSTREUUNGSMATRIX

die Matrix der *Intraklassenstreuung* für $\mathcal{C}_1$ ist. $\mathbf{S}_1/\sum_t r^t$ entspricht dem Schätzer für $\boldsymbol{\Sigma}_1$. Ähnlich dazu ist $s_2^2 = \boldsymbol{w}^T\mathbf{S}_2\boldsymbol{w}$ mit $\mathbf{S}_2 = \sum_t (1-r^t)(\boldsymbol{x}^t - \boldsymbol{m}_2)(\boldsymbol{x}^t - \boldsymbol{m}_2)^T$, und wir erhalten

$$s_1^2 + s_2^2 = \boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}$$

mit $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$ als Gesamtwert der Streuung innerhalb einer Klasse. Man beachte, dass $s_1^2 + s_2^2$ geteilt durch die Gesamtzahl an Stichproben der Varianz der im Pool zusammengefassten Daten entspricht. Gleichung 6.41 kann umgeformt werden zu

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T\mathbf{S}_B\boldsymbol{w}}{\boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}} = \frac{|\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)|^2}{\boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}}\,. \tag{6.45}$$

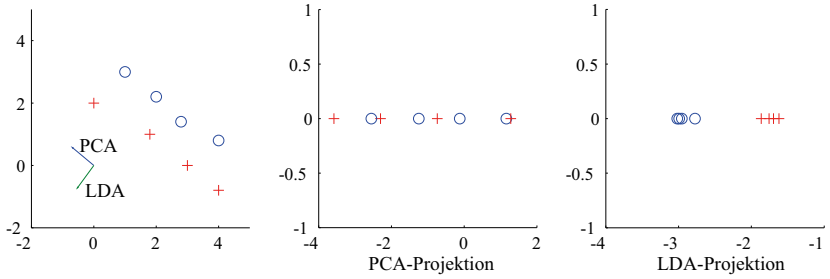


Abb. 6.11: Zweidimensionale synthetische Daten; dargestellt sind die durch PCA und LDA gefundenen Richtungen und die Projektionen entlang dieser Richtungen. LDA verwendet Klasseninformationen und liefert erwartungsgemäß eine viel bessere Klassenseparation.

Bilden wir die Ableitung von J hinsichtlich $\boldsymbol{w}$ und setzen sie gleich 0, so erhalten wir

$$\frac{\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}}\left(2(\boldsymbol{m}_1 - \boldsymbol{m}_2) - \frac{\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)}{\boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}}\mathbf{S}_W\boldsymbol{w}\right) = 0\,.$$

Da $\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_2)/\boldsymbol{w}^T\mathbf{S}_W\boldsymbol{w}$ eine Konstante ist, ergibt sich

$$\boldsymbol{w} = c\mathbf{S}_W^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)\,, \tag{6.46}$$

wobei c eine Konstante ist. Weil die Richtung für uns wichtig ist und nicht die Größenordnung, können wir einfach $c = 1$ setzen und $\boldsymbol{w}$ herausfinden.

Es sei noch einmal daran erinnert, dass im Falle von $p(\boldsymbol{x}|\mathcal{C}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$ eine lineare Diskriminante mit $\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ vorliegt, und wir erkennen, dass Fishers lineare Diskriminante optimal ist, wenn die Klassen normalverteilt vorliegen. Unter derselben Annahme kann auch ein Schwellwert w_0 berechnet werden, um die zwei Klassen voneinander zu trennen. Fishers lineare Diskriminante kann aber auch dann genutzt werden, wenn die Klassen nicht normalverteilt sind. Wir haben die Stichproben aus d Dimensionen auf eine projiziert, und eine beliebige Klassifikationsmethode kann im Anschluss angewendet werden. In Abbildung 6.11 sehen wir zweidimensionale synthetische Daten mit zwei Klassen. Wie wir sehen, ist die LDA-Richtung im Hinblick auf die anschließende Unterscheidung besser als die PCA-Richtung, was wir auch erwarten, da LDA Klasseninformationen verwendet.

Im Fall von $K > 2$ Klassen wollen wir die Matrix $\mathbf{W}$ finden, so dass

$$\boldsymbol{z} = \mathbf{W}^T\boldsymbol{x}\,, \tag{6.47}$$

wobei $\boldsymbol{z}$ k-dimensional ist und $\mathbf{W}$ gleich $d \times k$ ist. Die Matrix der Streuung innerhalb der Klasse für $\mathcal{C}_i$ ist

$$\mathbf{S}_i = \sum_t r_i^t(\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \tag{6.48}$$

mit $r_i^t = 1$, falls $\boldsymbol{x}^t \in \mathcal{C}_i$ und andernfalls 0. Die gesamte Streuung innerhalb von Klassen ist

$$\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i \,. \tag{6.49}$$

Wenn es also $K > 2$ Klassen gibt, berechnet sich die Streuung der Mittelwerte aus der Stärke ihrer Streuung um den gesamten Mittelwert

$$\boldsymbol{m} = \frac{1}{K} \sum_{i=1}^{K} \boldsymbol{m}_i \tag{6.50}$$

herum und die Interklassenstreuungsmatrix ergibt sich als

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T \tag{6.51}$$

mit $N_i = \sum_t r_i^t$. Die Interklassenstreuungsmatrix nach der Projektion ist $\mathbf{W}^T \mathbf{S}_B \mathbf{W}$ und die Matrix der Streuung innerhalb der Klassen ist nach der Projektion $\mathbf{W}^T \mathbf{S}_W \mathbf{W}$. Dies sind beides $k \times k$ Matrizen. Wir wollen, dass die erste Streuung möglichst hoch ist, das heißt, nach der Projektion sollten im neuen k-dimensionalen Raum die Klassenmittel so weit auseinander liegen wie möglich. Wir sind außerdem darum bemüht, dass die zweite Streuung möglichst klein ist, das heißt, nach der Projektion sollten die Stichproben von ein und derselben Klasse so nah wie möglich an ihrem Mittelwert liegen. Für eine Streuungsmatrix (oder eine Kovarianzmatrix) ist ein Maß der Ausbreitung die Determinante. Es sei daran erinnert, dass die Determinante das Produkt von Eigenwerten ist und dass ein Eigenwert die Varianz entlang seines Eigenvektors (der Eigenvektorkomponente) widerspiegelt. Somit interessiert uns die Matrix $\mathbf{W}$, welche den folgenden Term maximiert:

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|} \,. \tag{6.52}$$

Die größten Eigenvektoren von $\mathbf{S}_W^{-1} \mathbf{S}_B$ stellen die Lösung dar. $\mathbf{S}_B$ ist die Summe von K Matrizen mit Rang 1, sprich $(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$, und nur $K - 1$ von ihnen sind unabhängig. Somit hat $\mathbf{S}_B$ einen maximalen Rang von $K - 1$ und wir nutzen $k = K - 1$. Also definieren wir einen neuen niedrigeren, $(K - 1)$-dimensionalen Raum, in dem die Diskriminante dann zu errichten ist (siehe Abbildung 6.12). Zwar nutzt die LDA die Klassentrennbarkeit als Gütekriterium, jedoch kann jede beliebige Klassifikationsmethode in diesem neuen Raum genutzt werden, um die Diskriminanten zu schätzen.

Wir stellen fest, dass $\mathbf{S}_W$ invertierbar sein muss, damit die LDA anwendbar ist. Wenn dies nicht der Fall ist, können wir zunächst die PCA

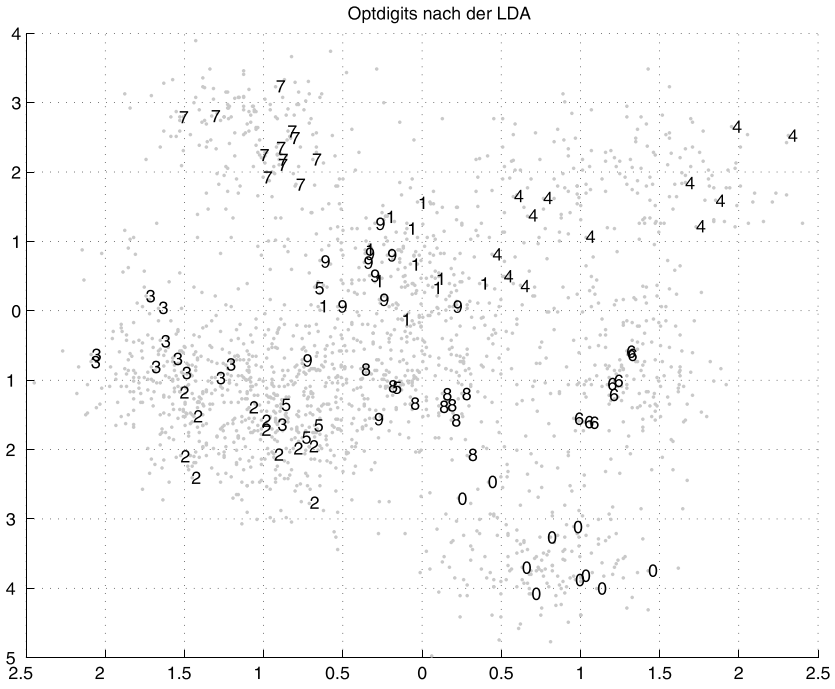


***Abb. 6.12*:** In den Raum der ersten beiden durch die LDA gefundenen Dimensionen eingetragene Optdigits-Daten. Beim Vergleich mit Abbildung 6.5 zeigt sich, dass die LDA – wie erwartet – zu einer besseren Klassentrennung führt als die PCA. Selbst in diesem zweidimensionalen Raum (es gibt 9 Dimensionen) können wir getrennte Wolken für die verschiedenen Klassen ausmachen.

anwenden, um die Singularität loszuwerden, und dann auf das Ergebnis die LDA anwenden. Dabei müssen wir allerdings sicherstellen, dass die PCA die Dimensionalität nicht so sehr reduziert, dass für die LDA nichts mehr übrig ist, womit sie arbeiten kann.

6.9 Kanonische Korrelationsanalyse

Bei allen Verfahren, die wir bisher diskutiert haben, haben wir angenommen, dass es eine einzige Datenquelle gibt, die uns eine einzige Menge an Beobachtungen liefert. Manchmal haben wir jedoch für ein und dasselbe Objekt oder Ereignis zwei Typen von Variablen. Bei der Spracherkennung zum Beispiel könnten wir zusätzlich zu den akustischen Informationen auch noch visuelle Informationen über die Lippenbewegungen während des Sprechens haben; beim Dokumentenretrieval haben wir Bilddaten und Textanmerkungen. Häufig sind die beiden Sätze von Variablen korreliert, und wir wollen diese Korrelation bei der Reduzierung der Dimensionali-

KANONISCHE KORRELATIONSANALYSE

tät zu einem gemeinsamen Raum berücksichtigen. Dies ist die Idee der *kanonischen Korrelationsanalyse* (abgekürzt CCA für engl. *canonical correlation analysis*) (Rencher 1995).

Angenommen, unsere Daten bestehen aus zwei Sätzen von Variablen, $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{y}^t\}_{t=1}^N$ mit $\boldsymbol{x}^t \in \mathbb{R}^d$ und $\boldsymbol{y}^t \in \mathbb{R}^e$. Beide Variablensätze sind Eingaben, und wir haben es hier mit einem unüberwachten Problem zu tun. Wenn eine Ausgabe für eine Klassifikation oder Regression gefordert ist, wird dies anschließend wie bei PCA gehandhabt (Abschnitt 6.3).

Die *kanonische Korrelation* ist die Summe der Korrelationen zwischen den Dimensionen $\boldsymbol{x}$ und den Dimensionen $\boldsymbol{y}$. Mit $\mathbf{S}_{xx} = \text{Cov}(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu}_x)^2]$ führen wir eine Notation ein, die die Kovarianzmatrix der Dimensionen $\boldsymbol{x}$ bezeichnet. Dies ist eine $d \times d$-Matrix, und sie entspricht der Matrix $\boldsymbol{\Sigma}$, die wir oft verwenden, beispielsweise bei PCA. Hier haben wir außerdem die $e \times e$-Matrix von $\boldsymbol{y}$, die wir mit $\mathbf{S}_{yy} = \text{Cov}(\boldsymbol{y})$ bezeichnen. Und wir haben die beiden Kreuz-Kovarianzmatrizen $\mathbf{S}_{xy} = \text{Cov}(\boldsymbol{x}, \boldsymbol{y}) = E[(\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)]$ (eine $d \times e$-Matrix) und $\mathbf{S}_{yx} = \text{Cov}(\boldsymbol{y}, \boldsymbol{x}) = E[(\boldsymbol{y} - \boldsymbol{\mu}_y)(\boldsymbol{x} - \boldsymbol{\mu}_x)]$ (eine $e \times d$-Matrix).

Wir sind interessiert an den beiden Vektoren $\boldsymbol{w}$ und $\boldsymbol{v}$, für die wir die maximale Korrelation erhalten, wenn wir $\boldsymbol{x}$ entlang $\boldsymbol{w}$ und $\boldsymbol{y}$ entlang $\boldsymbol{v}$ projizieren. Das heißt, wir wollen die folgende Größe maximieren:

$$\begin{aligned}\rho &= \text{Corr}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y}) = \frac{\text{Cov}(\boldsymbol{w}^T\boldsymbol{x}, \boldsymbol{v}^T\boldsymbol{y})}{\sqrt{\text{Var}(\boldsymbol{w}^T\boldsymbol{x})}\sqrt{\text{Var}(\boldsymbol{v}^T\boldsymbol{y})}} \\ &= \frac{\boldsymbol{w}^T\text{Cov}(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\text{Var}(\boldsymbol{x})\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\text{Var}(\boldsymbol{y})\boldsymbol{v}}} = \frac{\boldsymbol{w}^T\mathbf{S}_{xy}\boldsymbol{v}}{\sqrt{\boldsymbol{w}^T\mathbf{S}_{xx}\boldsymbol{w}}\sqrt{\boldsymbol{v}^T\mathbf{S}_{yy}\boldsymbol{v}}} . \end{aligned} \tag{6.53}$$

Gleichbedeutend damit können wir sagen, dass wir $\boldsymbol{w}^T\mathbf{S}_{xy}\boldsymbol{v}$ unter $\boldsymbol{w}^T\mathbf{S}_{xx}\boldsymbol{w} = 1$ und $\boldsymbol{v}^T\mathbf{S}_{yy}\boldsymbol{v} = 1$ maximieren wollen. Wenn wir dies, wie wir es bei PCA gemacht haben, in Lagrage-Ausdrücken schreiben, dann die Ableitungen nach $\boldsymbol{w}$ und $\boldsymbol{v}$ bilden und diese gleich 0 setzen, so sehen wir, dass $\boldsymbol{w}$ ein Eigenvektor von $\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}$ sein muss, und entsprechend muss $\boldsymbol{v}$ ein Eigenvektor von $\mathbf{S}_{yy}^{-1}\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\mathbf{S}_{xy}$ sein (Hardoon, Szedmak und Shawe-Taylor 2004).

Da wir an der Maximierung der Korrelation interessiert sind, wählen wir die beiden Eigenvektoren mit den größten Eigenwerten – nennen wir sie $\boldsymbol{w}_1$ und $\boldsymbol{v}_1$ – und der Grad der Korrelation ist gleich ihrem (gemeinsamen) Eigenwert λ_1. Die Eigenwerte von $\mathbf{AB}$ sind die gleichen wie die von $\mathbf{BA}$, solange $\mathbf{AB}$ und $\mathbf{BA}$ quadratisch sind, doch ihre Eigenvektoren sind nicht gleich: $\boldsymbol{w}_1$ ist d-dimensional, $\boldsymbol{v}_1$ dagegen e-dimensional.

Genauso, wie wir es bei PCA getan haben, können wir entscheiden, wie viele Paare von Eigenvektoren $(\boldsymbol{w}_i, \boldsymbol{v}_i)$ wir verwenden wollen, indem wir den relativen Wert des zugehörigen Eigenwerts betrachten:

$$\frac{\lambda_i}{\sum_{j=1}^s \lambda_j} .$$

Dabei ist $s = \min(d, e)$ der maximal mögliche Rang. Wir müssen ausreichend viele Eigenvektoren behalten, damit die Korrelation in den Daten erhalten bleibt.

Wenn wir k als Dimensionalität wählen, dann erhalten wir die *kanonischen Zufallsvariablen* durch Projektion der Trainingsdaten:

$$a_i^t = \boldsymbol{w}_i^T \boldsymbol{x}^t,\ b_i^t = \boldsymbol{v}_i^T \boldsymbol{y}^t,\ i = 1, \ldots, k\ . \tag{6.54}$$

Dies können wir in Matrixform schreiben als

$$\boldsymbol{a}^t = \mathbf{W}^T \boldsymbol{x}^t,\ \boldsymbol{b}^t = \mathbf{V}^T \boldsymbol{y}^t\ , \tag{6.55}$$

wobei $\mathbf{W}$ die $d \times k$-Matrix ist, deren Spalten die $\boldsymbol{w}_i$ sind, und $\mathbf{V}$ ist die $e \times k$-Matrix, deren Spalten die $\boldsymbol{v}_i$ sind (siehe Abbildung 6.13). Dieser Vektor aus Paaren (a_i, b_i) konstituiert nun unsere neue Darstellung, die von kleinerer Dimension ist und die wir zum Beispiel für die Klassifikation verwenden können. Diese neuen Merkmale sind nicht redundant: Die Werte von a_i sind unkorreliert und jedes a_i ist unkorreliert mit allen b_j, $j \neq i$.

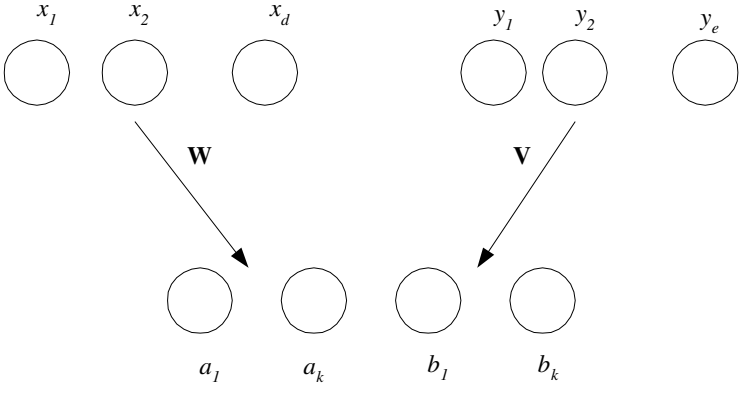


***Abb. 6.13*:** Die kanonische Korrelationsanalyse verwendet zwei Sätze von Variablen, $\boldsymbol{x}$ und $\boldsymbol{y}$, und projiziert sie jeweils so, dass die Korrelation nach der Projektion maximal ist.

Damit CCA sinnvoll ist, muss es zwischen den beiden Variablensätzen Abhängigkeiten geben. So gibt es beispielsweise beim Retrieval (Hardoon, Szedmak und Shawe-Taylor 2004) Abhängigkeiten: Das Wort „Himmel" ist assoziiert mit einer Menge blauer Farbe im Bild, und daher macht es hier Sinn, CCA zu verwenden. Doch das ist nicht immer der Fall. So könnten wir zum Beispiel bei einer Nutzerauthentifikation Unterschriften und Irisbilder haben, aber es gibt keinen Grund, irgendeine Abhängigkeit zwischen diesen anzunehmen. In diesem Fall wäre es besser, die Dimensionalitätsreduktion für die Unterschriften und die Irisbilder separat

durchzuführen und somit Abhängigkeiten zwischen Merkmalen innerhalb des gleichen Variablensatzes aufzudecken. Die Verwendung von CCA macht nur Sinn, wenn wir auch von Abhängigkeiten zwischen Merkmalen unterschiedlicher Variablensätze ausgehen können. In Rencher (1995) werden Tests diskutiert, mit denenn geprüft werden kann, ob $\mathbf{S}_{xy} = 0$ gilt, d. h. ob $\boldsymbol{x}$ und $\boldsymbol{y}$ unabhängig sind. Interessant ist folgender Zusammenhang zwischen CCA und Fishers LDA (Abschnitt 6.8): Sind $\boldsymbol{x}$ die beobachteten Variablen und sind Klassenlabels durch $\boldsymbol{y}$ als 1-aus-K-Code gegeben, dann findet CCA die gleiche Lösung wie LDA.

Bei der Faktorenanalyse geben wir eine generative Interpretation der Dimensionalitätsreduktion: Wir nehmen an, dass es verborgene Variablen $\boldsymbol{z}$ gibt, die zusammen die Ursache für die beobachteten Variablen $\boldsymbol{x}$ sind. Ähnlich können wir uns hier verborgene Variablen vorstellen, die $\boldsymbol{x}$ *und* $\boldsymbol{y}$ generieren. Tatsächlich können wir $\boldsymbol{a}$ und $\boldsymbol{b}$ zusammen als konstituierend für $\boldsymbol{z}$, die Repräsentation im latenten Raum, auffassen.

Es ist möglich, die kanonische Korrelationsanalyse so zu verallgemeinern, dass sie für mehr als zwei Sätze von Variablen anwendbar ist. Bach und Jordan (2005) geben eine probabilistische Interpretation von CCA, bei der mehr als zwei Variablensätze möglich sind.

6.10 Isomap

Die Hauptkomponentenanalyse, die wir in Abschnitt 6.3 diskutiert haben, ist anwendbar, wenn die Daten in einem linearen Unterraum liegen. Bei vielen Anwendungen ist das aber eventuell nicht der Fall. Betrachten wir beispielsweise eine Gesichtserkennung, bei der ein Gesicht durch ein zweidimensionales Bild aus 100×100 Pixeln gegeben ist. In diesem Fall ist jedes Gesicht ein Punkt in 10 000 Dimensionen. Nehmen wir an, wir machen eine Reihe von Aufnahmen, während eine Person ihren Kopf langsam von rechts nach links dreht. Die Sequenz der aufgezeichneten Gesichtsbilder folgt einer Trajektorie in diesem 10 000-dimensionalen Raum, und dies ist nicht linear. Betrachten wir nun die Gesichter vieler Personen. Die Trajektorien ihrer Gesichter, wenn sie jeweils die beschriebene Drehbewegung machen, definieren eine Mannigfaltigkeit in dem 10 000-dimensionalen Raum, und das ist es, was wir modellieren wollen. Die Ähnlichkeit zwischen zwei Gesichtern kann nicht einfach durch eine Summe von Pixeldifferenzen beschrieben werden, weshalb der Euklidische Abstand keine gute Metrik ist. Es kann sogar passieren, dass die Bilder zweier verschiedener Personen in der gleichen Pose einen kleineren Euklidischen Abstand haben als zwei Bilder der gleichen Person in unterschiedlichen Posen. Das ist natürlich nicht das, was wir wollen. Was zählen sollte, ist der Abstand entlang der Mannigfaltigkeit, der als *geodätischer Abstand* bezeichnet wird. Das *isometrische Merkmalsmapping,* kurz Isomap (Tenenbaum, de Silva und Langford 2000), schätzt diesen Abstand und wendet die multidimensionale Skalierung an (siehe

GEODÄTISCHER ABSTAND

ISOMAP

Abschnitt 6.7), wobei er für die Dimensionalitätsreduktion benutzt wird.

Isomap verwendet die geodätischen Abstände zwischen allen Paaren von Datenpunkten. Für benachbarte Punkte, die im Eingaberaum nahe beieinander liegen, kann der Euklidische Abstand verwendet werden. Für kleine Änderungen der Pose ist die Mannigfaltigkeit lokal linear. Für weit entfernte Punkte wird der geodätische Abstand durch die Summe der Abstände zwischen den Punkten entlang des Weges auf der Mannigfaltigkeit approximiert. Dazu wird ein Graph definiert, dessen Knoten den N Datenpunkten entsprechen. Seine Kanten verbinden benachbarte Punkte (das können diejenigen Punkte sein, deren Abstand zum betrachteten Knoten kleiner als ein gegebenes ϵ ist, oder auch die n nächstgelegenen), wobei Gewichte angewendet werden, die den Euklidischen Abständen entsprechen. Der geodätische Abstand zwischen zwei Punkten wird berechnet als die Länge des kürzesten Weges zwischen den zugehörigen Knoten. Für zwei nicht benachbarte Punkte müssen wir entlang des Weges über eine gewisse Anzahl von Zwischenpunkten gehen. Daher ist der Abstand der Abstand entlang der Mannigfaltigkeit, approximiert durch die Summe der lokalen Euklidischen Abstände (siehe Abbildung 6.14).

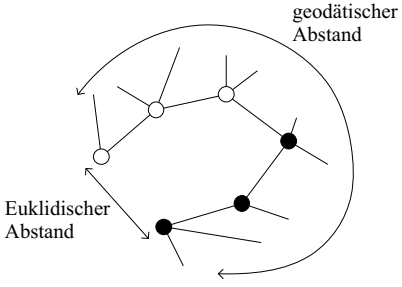


Abb. 6.14: Der geodätische Abstand wird entlang der Mannigfaltigkeit berechnet – im Unterschied zum Euklidischen Abstand, der diese Information nicht nutzt. Nach einer multidimensionalen Skalierung werden diese beiden Instanzen aus zwei Klassen auf weit voneinander entfernte Positionen in dem neuen Raum abgebildet, obwohl sie im ursprünglichen Raum nahe beieinander liegen.

Zwei Knoten r und s sind verbunden, wenn $\|\boldsymbol{x}^r - \boldsymbol{x}^s\| < \epsilon$ (wobei sichergestellt sein muss, dass der Graph verbunden ist), oder wenn $\boldsymbol{x}^s$ einer der n-Nachbarn von $\boldsymbol{x}^r$ ist (wobei sichergestellt sein muss, dass die Abstandsmatrix symmetrisch ist), und wir setzen die Kantenlänge auf $\|\boldsymbol{x}^r - \boldsymbol{x}^s\|$. Für jedes Paar von Knoten r und s ist d_{rs} die Länge des kürzesten Weges zwischen ihnen. Wir wenden dann MDS auf d_{rs} an, um die Dimensionalität auf k zu reduzieren, wobei die Merkmalseinbettung durch Beobachtung des Anteils der erklärten Varianz genutzt wird. Dies bewirkt, dass Knoten r und s, die im geodätischen Raum weit voneinander entfernt sind, auch in dem neuen, k-dimensionalen Raum weit voneinander entfernt platziert werden, selbst wenn sie im Sinne des Euklidischen Abstands in dem ursprünglichen, d-dimensionalen Raum nahe beieinander liegen.

Es ist offensichtlich, dass der Graphenabstand eine bessere Approximation liefert, wenn die Anzahl der Punkte zunimmt, wobei im Gegenzug

auch der Zeitaufwand wächst. Falls die Zeit kritisch ist, kann man sich auf Untermengen von „Orientierungspunkten“ beschränken, um den Algorithmus schneller zu machen. Der Parameter ϵ muss sorgfältig eingestellt werden – ist er zu klein, kann es passieren, dass es mehr als eine verbundene Komponente gibt, und ist er zu groß, könnten „Abkürzungen“ hinzugefügt werden, was die niedrigdimensionale Einbettung beschädigt (Balasubramanian et al. 2002).

Ein Problem, das bei Isomap wie auch MDS auftritt (da in beiden Fällen Merkmalseinbettung genutzt wird), besteht darin, dass N Punkte in einem niedrigdimensionalen Raum platziert werden, jedoch keine allgemeine Mapping-Funktion gelernt wird, die das Mapping eines neuen Testpunktes gestatten würde. Der neue Punkt sollte zu der Datenmenge hinzugefügt werden, um dann den gesamten Algorithmus noch einmal mit $N + 1$ Instanzen laufen zu lassen.

6.11 Lokal lineare Einbettung

LOKAL LINEARE EINBETTUNG

Die *lokal lineare Einbettung* (LLE) rekonstruiert die global nichtlineare Struktur aus lokal linearen Anpassungen (Roweis und Saul 2000). Die Idee ist, dass jeder loale Fleck der Mannigfaltigkeit linear approximiert werden kann und (wenn genug Daten gegeben sind) jeder Punkt als lineare, gewichtete Summe seiner Nachbarn geschrieben werden kann (wobei Nachbarschaft wieder entweder durch ein gegebene Zahl n von Nachbarn definiert ist, oder durch einen Abstandsschwellwert ϵ). Wenn ein Punkt $\boldsymbol{x}^r$ und seine Nachbarn $\boldsymbol{x}^s_{(r)}$ im Originalraum gegeben sind, dann findet man die Rekonstruktionsgewichte $\mathbf{W}_{rs}$, welche die Fehlerfunktion

$$\mathcal{E}^w(\mathbf{W}|\mathcal{X}) = \sum_r \left\|\boldsymbol{x}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{x}^s_{(r)}\right\|^2 \tag{6.56}$$

minimieren, unter Verwendung kleinster Quadrate, wobei $\mathbf{W}_{rr} = 0, \forall r$ und $\sum_s \mathbf{W}_{rs} = 1$.

LLE basiert auf der Idee, dass die Rekonstruktionsgewichte $\mathbf{W}_{rs}$ die intrinsischen geometrischen Eigenschaften der Daten widerspiegeln, von denen wir annehmen, dass sie auch für lokale Flecken der Mannigfaltigkeit gelten, d. h. für den neuen Raum, in den wir die Instanzen abbilden (siehe Abbildung 6.15). Der zweite Schritt von LLE besteht folglich darin, die Gewichte $\mathbf{W}_{rs}$ nun fix zu halten und die neuen Koordinaten $\boldsymbol{z}^r$ diejenigen Werte annehmen zu lassen, die nötig sind, um die durch die Gewichte gegebenen Zwischenpunktbedingungen zu erfüllen:

$$\mathcal{E}^z(\mathcal{Z}|\boldsymbol{W}) = \sum_r \left\|\boldsymbol{z}^r - \sum_s \mathbf{W}_{rs}\boldsymbol{z}^s\right\|^2. \tag{6.57}$$

Benachbarte Punkte im d-dimensionalen Originalraum sollten auch in dem neuen, k-dimensionalen Raum benachbart bleiben und eine ähnliche

Lage zueinander haben. Gleichung 6.57 kann umgeformt werden zu

$$\mathcal{E}^w(\mathcal{Z}|\boldsymbol{W}) = \sum_{r,s} \mathbf{M}_{rs}(\boldsymbol{z}^r)^T \boldsymbol{z}^s \tag{6.58}$$

mit

$$\mathbf{M}_{rs} = \delta_{rs} - \mathbf{W}_{rs} - \mathbf{W}_{sr} + \sum_i \mathbf{W}_{ir}\mathbf{W}_{is} \; . \tag{6.59}$$

$\mathbf{M}$ ist eine dünn besetzte Matrix (nur ein kleiner Prozentsatz der Datenpunkte sind Nachbarn eines bestimmten Datenpunkts: $n \ll N$), symmetrisch und positiv semidefinit. Wie bei anderen Verfahren zur Dimensionalitätsreduktion fordern wir, dass die Daten um den Ursprung zentriert sind, also $E[\boldsymbol{z}] = 0$, und dass die neuen Koordinaten unkorreliert und von Einheitslänge sind: $\mathrm{Cov}(\boldsymbol{z}) = \mathbf{I}$. Die Lösung von Gleichung 6.58 unter diesen beiden Nebenbedingungen ist gegeben durch die $k+1$ Eigenvektoren mit den kleinsten Eigenwerten. Wir ignorieren den niedrigsten, und die anderen k Eigenvektoren liefern uns die neuen Koordinaten.

Da die n Nachbarn einen Raum der Dimension $n-1$ aufspannen (man braucht die Abstände zu drei Punkten, um die Position in zwei Dimensionen eindeutig beschreiben zu können), kann LLE die Dimension bis auf $k \leq n-1$ reduzieren. Es wurde beobachtet (Saul und Roweis 2003), dass ein gewisser Margin zwischen k und n notwendig ist, um eine gute Einbettung zu erhalten. Man beachte, dass es für kleine n (oder ϵ) passieren kann, dass der Graph (der durch Verbinden jeder Instanz mit ihren Nachbarn konstruiert wurde) nicht mehr verbunden ist, so dass es notwendig werden kann, LLE separat auf jeder Komponente laufen zu lassen, um in

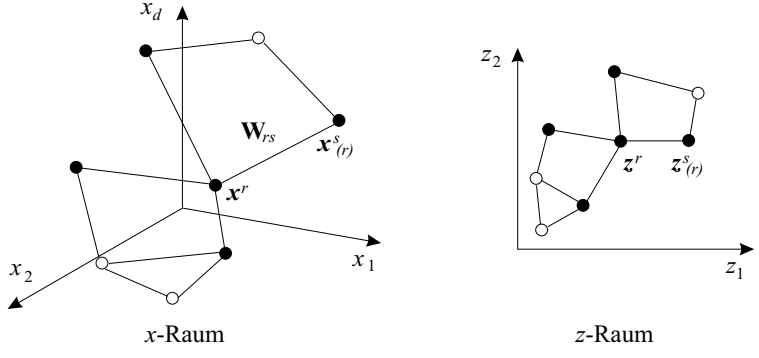


Abb. 6.15: Bei der lokal linearen Einbettung werden zunächst die Nebenbedingungen im Originalraum gelernt und die Punkte dann unter Beachtung dieser Nebenbedingungen in dem neuen Raum platziert. Zum Lernen der Nebenbedingungen werden die nächsten Nachbarn benutzt (dargestellt als durchgezogene Linien), aber auch Nachbarn zweiter Ordnung (gestrichelte Linien).

verschiedenenen Teilen des Eingaberaums separate Mannigfaltigkeiten zu finden. Wenn n (oder ϵ) dagegen groß ist, können manche Nachbarn so weit entfernt sein, dass die Annahme der lokalen Linearität nicht mehr gilt, was die Einbettung beschädigen kann. Es ist möglich, auf der Basis von Vorwissen in verschiedenen Teilen des Eingaberaums verschiedene n (oder ϵ) zu verwenden, doch wie man das genau macht, ist eine offene Frage (Saul und Roweis 2003).

Wie bei Isomap ist die Lösung auch bei LLE der Satz von neuen Koordinaten für die N Punkte, doch es gibt dabei kein Lernen des Mappings, so dass es nicht möglich ist, $\boldsymbol{z}'$ für ein neues $\boldsymbol{x}'$ zu finden. Es gibt zwei Lösungen für dieses Problem:

1. Mit der gleichen Idee kann man die n Nachbarn von $\boldsymbol{x}'$ in dem d-dimensionalen Originalraum finden und zunächst die Rekonstruktionsgewichte $\boldsymbol{w}_j$ lernen, die

$$\mathcal{E}^w(\mathbf{W}|\mathcal{X}) = \left\|\boldsymbol{x}' - \sum_s \boldsymbol{w}_s \boldsymbol{x}^s\right\|^2 \tag{6.60}$$

   minimieren. Die Gewichte verwendet man dann, um $\boldsymbol{z}'$ in dem neuen k-dimensionalen Raum zu rekonstruieren:

$$\boldsymbol{z}' = \sum_s \boldsymbol{w}_s \boldsymbol{z}^s \,. \tag{6.61}$$

   Dieser Ansatz kann auch verwendet werden, um auf der Basis einer Isomap-Lösung (oder MDS) zu interpolieren. Der Nachteil ist allerdings, dass die gesamte Menge der $\{\boldsymbol{x}^t, \boldsymbol{z}^t\}_{t=1}^N$ gespeichert werden muss.

2. Indem man $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{z}^t\}_{t=1}^N$ als Trainingsmenge verwendet, kann man jeden Regressor $\boldsymbol{g}(\boldsymbol{x}^t|\theta)$ – beispielsweise ein mehrlagiges Perzeptron (Kapitel 11) – dazu trainieren, $\boldsymbol{z}^t$ aus $\boldsymbol{x}^t$ zu approximieren, wobei der Parameter θ so trainiert wird, dass er den Regressionsfehler

$$\mathcal{E}(\theta|\mathcal{X}) = \sum_t \left\|\boldsymbol{z}^t - \boldsymbol{g}(\boldsymbol{x}^t|\theta)\right\|^2 \tag{6.62}$$

   minimiert. Nachdem das Training abgeschlossen ist, können wir $\boldsymbol{z}' = \boldsymbol{g}(\boldsymbol{x}'|\theta)$ berechnen. Das Modell $\boldsymbol{g}(\cdot)$ muss sorgfältig gewählt werden, damit es in der Lage ist, das Mapping zu lernen. Es kann sein, dass es kein eindeutiges Optimum mehr gibt, was all die üblichen Probleme im Zusammenhang mit der Minimierung mit sich bringt, also die Frage nach der Initialisierung, das Problem lokaler Optima, Konvergenzeigenschaften usw.

Sowohl bei Isomap als auch bei LLE gibt es lokale Informationen, die sich über die Nachbarn fortpflanzen, woraus sich eine globale Lösung ergibt.

Bei Isomap ist der geodätische Abstand die Summe der lokalen Abstände. Bei LLE berücksichtigt die finale Optimierung der Platzierung von $\boldsymbol{z}^t$ alle lokalen Werte von $\mathbf{W}_{rs}$. Nehmen wir an, a und b sind Nachbarn und b und c sind Nachbarn. Obwohl a und c möglicherweise keine Nachbarn sind, gibt es eine Abhängigkeit zwischen a und c, die entweder durch den Graph als $d_{ac} = d_{ab} + d_{bc}$ oder die Gewichte $\mathbf{W}_{ab}$ und $\mathbf{W}_{bc}$ beschrieben werden kann. Bei beiden Algorithmen findet man die global nichtlineare Organisation durch Integration lokal linearer Nebenbedingungen, die sich partiell überlappen.

6.12 Laplacesche Eigenmaps

Betrachten wir die Dateninstanz $\boldsymbol{x}^r \in \mathbb{R}^d$, $r = 1, \ldots, N$ und ihre Projektion $\boldsymbol{z}^r \in \mathbb{R}^k$. Es sei ein Ähnlichkeitsmaß B_{rs} zwischen Paaren von Instanzen gegeben, das vielleicht in irgendeinem hochdimensionalen Raum so berechnet wurde, dass es sein Maximum annimmt, wenn r und s gleich sind, und das umso kleiner wird, je unähnlicher r und s sich werden. Weiter nehmen wir an, dass der kleinstmögliche Wert von B_{rs} null ist und dass das Maß symmetrisch ist, d. h., es gilt $B_{rs} = B_{sr}$ (Belkin und Nyogi 2003). Das Ziel ist die folgende Minimierung:

$$\min \sum_{r,s} \|\boldsymbol{z}^r - \boldsymbol{z}^s\|^2 B_{rs} \,. \tag{6.63}$$

Zwei Instanzen, die ähnlich sein sollen, also ein Paar r und s mit großem B_{rs}, sollten in dem neuen Raum nahe beieinander liegen; folglich ist der Abstand zwischen $\boldsymbol{z}^r$ und $\boldsymbol{z}^s$ klein. Je unähnlicher sie sich sind, umso weniger interessieren wir uns für ihre relative Position in dem neuen Raum. Die B_{rs} werden im Originalraum berechnet. Wenn wir zum Beispiel das Skalarprodukt verwenden, arbeitet das Verfahren ähnlich wie die multidimensionale Skalierung:

$$B_{rs} = (\boldsymbol{x}^r)^T \boldsymbol{x}^s \,.$$

Laplacesche Eigenmaps

Doch bei *Laplaceschen Eigenmaps* interessieren wir uns, ähnlich wie bei Isomap und LLE, für Ähnlichkeiten nur lokal (Belkin and Nyogi 2003). Wir definieren eine Nachbarschaft entweder durch einen maximalen Abstand ϵ zwischen $\boldsymbol{x}^r$ und $\boldsymbol{x}^s$ oder durch die k-nächsten-Nachbarn; außerhalb dieser Nachbarschaft setzen wir B_{rs} gleich 0. In der Nachbarschaft verwenden wir den Gauß-Kernel, um den Euklidischen Abstand in einen Ähnlichkeitswert zu überführen:

$$B_{rs} = \exp\left[-\frac{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|^2}{2\sigma^2}\right] \,.$$

Dabei ist σ ein nutzerdefinierter Wert. $\mathbf{B}$ definiert einen gewichteten Graph.

Im Falle $k = 1$ (wir reduzieren die Dimensionalität auf 1) können wir Gleichung 6.63 wie folgt umformen:

$$\begin{aligned}
&\min \ \frac{1}{2}\sum_{r,s}(z_r - z_s)^2 B_{rs} \\
&= \frac{1}{2}\left(\sum_{r,s} B_{rs} z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_{r,s} B_{rs} z_s^2\right) \\
&= \frac{1}{2}\left(\sum_{r} d_r z_r^2 - 2\sum_{r,s} B_{rs} z_r z_s + \sum_{s} d_s z_s^2\right) \\
&= \sum_{r} d_r z_r^2 - \sum_{r}\sum_{s} B_{rs} z_r z_s \\
&= \boldsymbol{z}^T \mathbf{D} \boldsymbol{z} - \boldsymbol{z}^T \mathbf{B} \boldsymbol{z} \,. \qquad (6.64)
\end{aligned}$$

Hierbei ist $d_r = \sum_s B_{rs}$. $\mathbf{D}$ ist die Diagonalenmatrix der d_r und $\boldsymbol{z}$ ist der N-dimensionale Spaltenvektor, dessen r-te Komponente, und z_r, die neue Koordinate für $\boldsymbol{x}^r$ ist. Wir definieren den Laplace-Operator des Graphen

$$\mathbf{L} = \mathbf{D} - \mathbf{B} \qquad (6.65)$$

und unser Ziel ist es, $\boldsymbol{z}^T \mathbf{L} \boldsymbol{z}$ zu minimieren. Für eine eindeutige Lösung definieren wir $\|\boldsymbol{z}\| = 1$. Wie bei der Merkmalseinbettung erhalten wir die Koordinaten in dem neuen Raum direkt, also ohne Projektion, und man kann zeigen, dass $\boldsymbol{z}$ ein Eigenvektor von $\mathbf{L}$ sein sollte. Da wir minimieren wollen, wählen wir den Eigenvektor mit dem kleisten Eigenwert. Allerdings ist zu beachten, dass es mindestens einen Eigenvektor mit dem Eigenwert 0 gibt, und dieser wird ignoriert. Die Komponenten dieses Eigenvektors sind alle gleich: $\boldsymbol{c} = (1/\sqrt{N})\mathbf{1}^T$. Der zugehörige Eigenwert ist null, da

$$\mathbf{L}\boldsymbol{c} = \mathbf{D}\boldsymbol{c} - \mathbf{B}\boldsymbol{c} = 0 \,.$$

In der Diagonale von $\mathbf{D}$ stehen Zeilensummen, und das Skalarprodukt aus einer Zeile von $\mathbf{B}$ und $\mathbf{1}$ bildet ebenfalls eine gewichtete Summe. Damit 6.64 bei nichtnegativen B_{ij} null wird, müssen z_i und z_j für alle Paare i, j gleich sein, und da die Norm 1 sein soll, müssen sie alle $1/\sqrt{N}$ sein. Also müssen wir den Eigenvektor mit Eigenwert 0 weglassen, und wenn wir die Dimensionalität auf $k > 1$ reduzieren wollen, müssen wir das nächste k nehmen.

Die Laplacesche Eigenmap ist ein Verfahren der Merkmalseinbettung, d. h., wir finden die Koordinaten im neuen Raum direkt und haben kein explizites Modell für das Mapping, das wir später für neue Instanzen verwenden könnten.

Vergleichen wir Gleichung 6.63 mit Gleichung 6.37 (Sammon-Stress in MDS). Hier wird die Ähnlichkeit im Originalraum implizit durch B_{rs}

dargestellt, während sie in MDS explizit als $\|\boldsymbol{x}^r - \boldsymbol{x}^s\|$ geschrieben wird. Ein weiterer Unterschied ist, dass wir bei MDS alle Paare auf Ähnlichkeit testen, während hier die Nebenbedingungen lokal sind (sie pflanzen sich dann fort, weil sich die lokalen Nachbarschaften teilweise überlappen – wie in Isomap und LLE).

Für die vierdimensionalen Irisdaten sind die Ergebnisse von MDS und Laplaceschen Eigenmaps nach der Projektion auf zwei Dimensionen in Abbildung 6.16 dargestellt. MDS ist hier äquivalent mit PCA, während die Laplacesche Eigenmap, wie wir sehen, ähnliche Instanzen in nahe beieinander gelegene Punkte des neuen Raums abbildet. Aus diesem Grund bietet das Verfahren eine gute Möglichkeit, die Daten vor einer Clusteranalyse aufzubereiten. Die spektrale Clusteranalyse, die wir in Abschnitt 7.7 behandeln, verwendet diesen Ansatz.

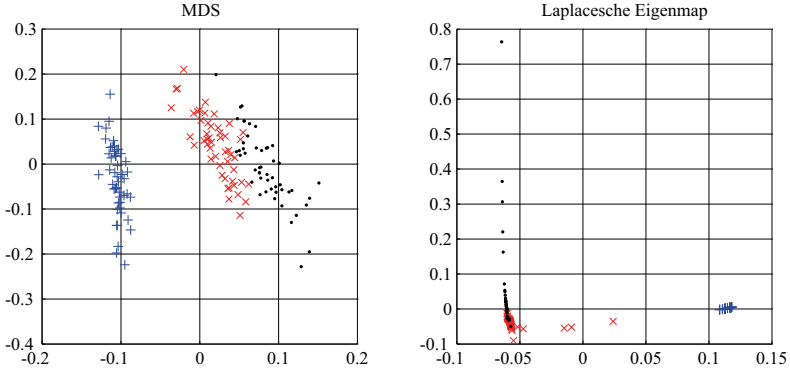


Abb. 6.16: Auf zwei Dimensionen reduzierte Irisdaten, die mit multidimensionaler Skalierung bzw. Laplaceschen Eigenmaps erhalten wurden. Letzteres führt zu einer viel dichteren Platzierung ähnlicher Instanzen.

6.13 Anmerkungen

Die Auswahl der Teilmengen bei der Regression wird bei Miller (1990) diskutiert. Die Vorwärts- und Rückwärtssuchprozeduren, die wir besprochen haben, sind lokale Suchprozeduren. Fukunaga und Narendra (1977) haben eine Prozedur des Verzweigens und Begrenzens vorgeschlagen („branch and bound“). Es ist auch möglich, allerdings mit deutlich höheren Kosten verbunden, eine stochastischen Methode wie Simulated Annealing oder genetische Algorithmen zu verwenden, um in größerem Umfang im Suchraum zu suchen.

Es gibt auch *Filteralgorithmen* für die Merkmalsselektion, bei denen heuristische Maße verwendet werden, um die „Relevanz“ eines Merkmals in einer Verarbeitungsstufe zu berechnen, ohne tatsächlich den Lerner

zu verwenden. Im Falle der Klassifikation kann man etwa, anstatt einen Klassifikator zu verwenden und ihn in jedem Schritt zu testen, auch ein Maß für die Trennschärfe wie das bei der linearen Diskriminanzanalyse gewählte nutzen, um zu messen, wie gut der neue Raum die Klassen voneinander trennt (McLachlan 1992). In Anbetracht sinkender Rechenkosten ist es am besten, den Lerner in die Schleife aufzunehmen, da es keine Garantie dafür gibt, dass die von dem Filter verwendete Heuristik zu der Verzerrung des Lerners passt, der die Merkmale verwendet. Keine Heuristik kann die tatsächliche Validierungsgenauigkeit ersetzen. Eine Übersicht über Verfahren der Merkmalsselektion ist in Guyon und Elisseeff (2003) zu finden.

Projektionsmethoden arbeiten mit numerischen Eingaben, und diskrete Variablen sollten durch 0/1-Dummy-Variablen repräsentiert werden, wohingegen die Teilmengenselektion diskrete Eingaben direkt verarbeiten kann. Das Bestimmen von Eigenvektoren und Eigenwerten ist nicht sehr kompliziert, und es ist als Standardprozedur in jedem Softwarepaket zur linearen Algebra enthalten. Die Faktorenanalyse wurde durch den britischen Psychologen Charles Spearman eingeführt, um den einen Intelligenzfaktor herauszufinden, welcher die Korrelation zwischen bei verschiedenen Intelligenztests erzielten Ergebnissen erklärt. Die Existenz solch eines einzelnen Faktors, genannt g, wird kontrovers diskutiert. Mehr Informationen zur multidimensionalen Skalierung finden sich bei Cox und Cox (1994).

Die hier behandelten Projektionsmethoden sind Batch-Prozeduren, und zwar insofern, als sie voraussetzen, dass die gesamte Stichprobe gegeben sein muss, bevor die Projektionsrichtungen gefunden werden. Mao und Jain (1995) diskutieren online Prozeduren zur Durchführung der PCA und LDA, bei denen Instanzen eine nach der anderen gegeben und Aktualisierungen bei Ankunft neuer Instanzen vorgenommen werden. Eine weitere Möglichkeit für die Durchführung einer nichtlinearen Projektion bietet sich, wenn der Schätzer beim Sammon-Mapping als nichtlineare Funktion gewählt wird, beispielsweise ein mehrlagiges Perzeptron (Abschnitt 11.11) (Mao und Jain 1995). Es ist ebenfalls möglich, jedoch deutlich schwerer, eine nichtlineare Faktorenanalyse durchzuführen. Wenn die Modelle nichtlinear sind, ist es schwierig, das richtige nichtlineare Modell zu finden. Es ist außerdem nötig, komplizierte Optimierungs- und Approximationsmethoden zu verwenden, um die Modellparameter zu bestimmen.

Laplacesche Eigenmaps verwenden die Idee der Merkmalseinbettung in der Weise, dass gegebene paarweise Ähnlichkeiten erhalten bleiben. Auf der gleichen Idee basieren Kernel-Maschinen, bei denen paarweise Ähnlichkeiten durch eine Kernel-Funktion gegeben sind. In Kapitel 13 beschäftigen wir uns mit „Kernel-Versionen“ von PCA, LDA und CCA. Ebenso, wie wir die polynomiale Regression implementiert haben, indem wir die lineare Regression um Terme höherer Ordnung als zusätzliche Eingaben erweitert haben (Abschnitt 5.8), können wir eine nichtlineare Dimensionalitätsreduktion durch Mapping auf einen neuen Raum konstru-

ieren, wobei wir nichtlineare Basisfunktionen verwenden. Das ist die Idee hinter Kernel-Verfahren, mit denen wir bei der Ähnlichkeitsberechnung weiter kommen als mit Skalarprodukten oder Euklidischen Abständen.

Verfahren zum Zerlegen von Matrizen sind im Zusammenhang mit Big-Data-Anwendungen weit verbreitet, da sie es gestatten, eine große Datenmatrix durch kleinere Matrizen zu beschreiben. Angewendet wird dies u. a. in *Empfehlungssystemen*, in denen es zum Beispiel Millionen von Filmen und Millionen von Nutzern gibt. Die Matrixelemente sind hier die Nutzerbewertungen, wobei die meisten Elemente fehlen und das Ziel gerade darin besteht, die Fehlstellen aufzufüllen, um anhand der vorhergesagten Werte Empfehlungen zu generieren (Koren, Bell und Volinsky 2009).

Empfehlungssysteme

Es gibt einen Kompromiss zwischen Merkmalsextraktion und der Entscheidungsfindung. Wenn der Merkmalsextraktor gut ist, wird die Aufgabe des Klassifikators (oder Regressors) eher trivial, beispielsweise wenn der Klassencode als ein neues Merkmal aus den existierenden Merkmalen extrahiert wird. Andererseits, wenn der Klassifikator gut genug arbeitet, dann gibt es keinen Bedarf für die Merkmalsextraktion; seine automatische Merkmalsselektion oder -kombination wird intern vorgenommen. Wir bewegen uns zwischen diesen beiden idealen Welten.

Es gibt Algorithmen, die intern eine Merkmalsselektion durchführen, wenn auch in beschränktem Maße. Entscheidungsbäume (Kapitel 9) führen während der Generierung des Entscheidungsbaumes eine Merkmalsselektion durch, und in mehrlagigen Perzeptronen (Kapitel 11) läuft eine nichtlineare Merkmalsextraktion in den verborgenen Knoten. Wir erwarten weitere Entwicklungen in dieser Richtung durch Kopplung der Merkmalsextraktion und dem späteren Schritt der Klassifikation/Regression.

6.14 Übungen

1. Wie kann die neue Diskriminante bei der Teilmengenselektion unter Annahme normalverteilter Klassen beim Hinzufügen oder Entfernen einer Variable schnell berechnet werden? Wie kann zum Beispiel der neue Wert $\mathbf{S}_{neu}^{-1}$ aus $\mathbf{S}_{alt}^{-1}$ berechnet werden?
2. Implementieren sie die PCA unter Nutzung der Optdigits aus der UCI-Datensammlung. Rekonstruieren Sie die Schriftzeichenbilder für eine verschiedene Zahl an Eigenvektoren und berechnen Sie den Rekonstruktionsfehler (Gleichung 6.12).
3. Tragen Sie die Landkarte Ihres Bundeslandes/Staates unter Zuhilfenahme von MDS ab, wenn die Entfernungen auf dem Landweg als Eingabe gegeben sind.
4. Wenn beim Sammon-Mapping eine lineare Abbildung vorliegt, also $g(\boldsymbol{x}|\mathbf{W}) = \mathbf{W}^T\boldsymbol{x}$, wie kann $\mathbf{W}$ berechnet werden, so dass sich der Sammon-Stress minimiert?

5. Abbildung 6.11 zeigt synthetische zweidimensionale Daten, für die LDA ein besseres Ergebnis liefert als PCA. Skizzieren Sie eine ähnliche Datenmenge, für die PCA und LDA die gleiche Richtung finden. Skizzieren Sie eine weitere Datenmenge, für die weder PCA noch LDA eine gute Richtung finden.

6. Wiederholen Sie Übung 3, diesmal jedoch unter Verwendung von Isomap, wobei zwei Städte nur dann verbunden sind, wenn es eine direkte Straße zwischen ihnen gibt, die nicht durch irgendeine andere Stadt geht.

7. In Isomap können wir anstelle des Euklidischen Abstands auch den Mahalanobis-Abstand zwischen benachbarten Punkten verwenden. Welche Vor- und Nachteile hat dieser Ansatz?

8. Multidimensionale Skalierung funktioniert, solange wir die paarweisen Abstände zwischen den Objekten haben. Wir müssen die Objekte noch nicht einmal tatsächlich durch Vektoren darstellen, solange wir ein Ähnlichkeitsmaß haben. Können Sie hierfür ein Beispiel angeben?

   Lösung: Angenommen, unsere Datenbasis besteht aus Dokumenten. Wir bezeichnen die Anzahl der Begriffe, die sowohl in Dokument r, als auch in Dokument s vorkommen, mit d_{rs} und bilden diese Dokumente mithilfe von MDS auf einen Raum von geringerer Dimension ab, so dass wir sie zum Beispiel visualisieren und hinsichtlich ihrer Struktur prüfen können. Beachten Sie, dass wir die Anzahl der gemeinsamen Begriffe zählen können, ohne die Dokumente explizit durch Vektoren darstellen zu müssen, etwa mithilfe der Bag-of-Words-Repräsentation.

9. Wie können wir Klasseninformationen in Isomap oder LLE einbauen, so dass Instanzen der gleichen Klasse im neuen Raum in dicht benachbarten Punkten abgebildet werden?

   Lösung: Wir können einen Strafterm hinzufügen, wobei wir Abstände für Instanzen berechnen, die zu unterschiedlichen Klassen gehören. MDS wird dann Instanzen der gleichen Klasse in benachbarte Punkte abbilden.

10. Wie können wir bei der Faktorenanalyse die noch fehlenden Faktoren finden, wenn wir einige Faktoren bereits kennen?

    Lösung: Wenn wir bereits einige Faktoren kennen, dann können wir ihre Ladungen durch Regression finden und dann ihren Einfluss aus den Daten entfernen. Was wir dann erhalten, wird nicht durch jene Faktoren erklärt, und wir können nun nach zusätzlichen Faktoren suchen, die den Rest erklären.

11. Diskutieren Sie eine Anwendung, bei der es verborgene (nicht zwangsläufig lineare) Faktoren gibt und für die zu erwarten ist, dass die Faktorenanalyse gut funktioniert.

LÖSUNG: Ein Beispiel sind die Prädikate bei akademischen Graden, die die Absolventen einer Universität verliehen bekommen. Das Prädikat, das ein Student oder eine Studentin nach einer Reihe von Kursen bekommt, hängt von einigen verborgenen Faktoren ab, etwa von seiner bzw. ihrer Begabung für das jeweilige Fach, von der Zeit, die er oder sie für das Studium aufbringen kann, vom Komfort der Unterkunft und anderem mehr.

6.15 Literaturangaben

Balasubramanian, M., E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford. 2002. „The Isomap Algorithm and Topological Stability.“ *Science* 295: 7.

Bach, F., and M. I. Jordan. 2005. *A Probabilistic Interpretation of Canonical Correlation Analysis.* Technical Report 688, Department of Statistics, University of California, Berkeley.

Belkin, M., and P. Niyogi. 2003. „Laplacian Eigenmaps for Dimensionality Re- duction and Data Representation.“ *Neural Computation* 15: 1373–1396.

Chatfield, C., and A. J. Collins. 1980. *Introduction to Multivariate Analysis.* London: Chapman and Hall.

Cox, T. F., and M. A. A. Cox. 1994. *Multidimensional Scaling.* London: Chapman and Hall.

Flury, B. 1988. *Common Principal Components and Related Multivariate Models.* New York: Wiley.

Fukunaga, K., and P. M. Narendra. 1977. „A Branch and Bound Algorithm for Feature Subset Selection.“ *IEEE Transactions on Computers* C-26: 917–922.

Guyon, I., and A. Elisseeff. 2003. „An Introduction to Variable and Feature Selection.“ *Journal of Machine Learning Research* 3: 1157–1182.

Hardoon, D. R., S. Szedmak, J. Shawe-Taylor. 2004. „Canonical Correlation Analysis: An Overview with Application to Learning Methods.“ *Neural Computation* 16: 2639–2664.

Hastie, T. J., R. J. Tibshirani, and A. Buja. 1994. „Flexible Discriminant Analysis by Optimal Scoring.“ *Journal of the American Statistical Association* 89: 1255–1270.

Kohavi, R., and G. John. 1997. „Wrappers for Feature Subset Selection.“ *Artificial Intelligence* 97: 273–324.

Koren, Y., R. Bell, and C. Volinsky. 2009. „Matrix Factorization Techniques for Recommender Systems." *IEEE Computer* 42 (8): 30–37.

Landauer, T. K., D. Laham, and M. Derr. 2004. „From Paragraph to Graph: Latent Semantic Analysis for Information Visualization." *Proceedings of the National Academy of Sciences* 101 (suppl. 1): 5214–5219.

Lee, D. D., and H. S. Seung. 1999. „Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature* 401 (6755): 788–791.

Mao, J., and A. K. Jain. 1995. „Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Miller, A. J. 1990. *Subset Selection in Regression.* London: Chapman and Hall.

Pudil, P., J. Novovičová, and J. Kittler. 1994. „Floating Search Methods in Feature Selection." *Pattern Recognition Letters* 15: 1119–1125.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Roweis, S. T., and L. K. Saul. 2000. „Nonlinear Dimensionality Reduction by Locally Linear Embedding." *Science* 290: 2323–2326.

Saul, K. K., and S. T. Roweis. 2003. „Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds." *Journal of Machine Learning Research* 4: 119–155.

Strang, G. 2006. *Linear Algebra and Its Applications,* 4th ed. Boston: Cengage Learning.

Tenenbaum, J. B., V. de Silva, and J. C. Langford. 2000. „A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science* 290: 2319–2323

Tipping, M. E., and C. M. Bishop. 1999. „Probabilistic Principal Components Analysis." Technical Report NCRG/97/010, *Journal of the Royal Statistical Society Series B* 61: 611–622.

Turk, M., and A. Pentland. 1991. „Eigenfaces for Recognition." *Journal of Cognitive Neuroscience* 3: 71–86.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

7 Clusteranalyse

Beim parametrischen Ansatz nahmen wir an, dass die Stichprobe aus einer bekannten Verteilung entstammt. In Fällen, bei denen eine solche Annahme unhaltbar ist, entschärfen wir diese Voraussetzung und nutzen einen semiparametrischen Ansatz, der es erlaubt, eine Mischung aus Verteilungen für die Schätzung der Eingabestichprobe zu nutzen. Clustermethoden gestatten es, die Parameter solch einer Mischung anhand der Daten zu erlernen. Zusätzlich zur probabilistischen Modellierung diskutieren wir die Vektorquantisierung, die spektrale Clusteranalyse und die hierarchische Clusteranalyse.

7.1 Einführung

In Kapitel 4 und 5 haben wir die parametrische Methode zur Dichteschätzung diskutiert, bei der wir annahmen, dass die Stichprobe $\mathcal{X}$ aus einer parametrischen Familie gezogen wurde, beispielsweise aus der Gaußschen. Bei der parametrischen Klassifikation entspricht dies der Annahme einer gewissen Dichte für die Klassendichten $p(\boldsymbol{x}|\mathcal{C}_i)$. Der Vorteil eines jeden parametrischen Ansatzes besteht darin, dass sich bei einem vorliegendem Modell das Problem auf die Schätzung einer geringen Anzahl an Parametern reduziert, welche im Fall der Dichteschätzung die ausreichenden Statistiken der Dichte darstellen, beispielsweise den Mittelwert und die Kovarianz im Falle der Gaußschen Dichten.

Zwar werden parametrische Ansätze recht häufig genutzt, jedoch kann die Annahme eines rigiden parametrischen Modells eine Ursache für Verzerrung bei vielen Anwendungen darstellen, bei denen diese Annahme unhaltbar ist. Daher benötigen wir flexiblere Modelle. Im Detail entspricht die Annahme einer Gaußschen Dichte der Annahme, dass die Stichprobe, beispielsweise Instanzen einer Klasse, eine einzelne Gruppe im d-dimensionalen Raum formt, und, wie wir in Kapitel 5 sahen, sich das Zentrum und die Form dieser Gruppe durch den Mittelwert beziehungsweise die Kovarianz ergibt.

Bei vielen Anwendungen ist die Stichprobe jedoch keine einzelne Gruppe; es kann durchaus mehrere Gruppen geben. Man nehme den Fall der optischen Schriftzeichenerkennung. Es gibt zwei Arten, die Ziffer 7 zu schreiben; die amerikanische Schreibweise ist „7“, wohingegen bei der europäischen Schreibweise noch ein horizontaler Balken in der Mitte

hinzugefügt wird (um sie von der europäischen „1“ zu unterscheiden, bei deren handschriftlicher Version der kleine obere Anstrich beibehalten wird). In solch einem Szenario, wenn die Stichprobe Beispiele von beiden Kontinenten enthält, sollte die Klasse für das Zeichen 7 als die Disjunktion von zwei Gruppen dargestellt werden. Wenn jede dieser beiden Gruppen durch eine Gaußverteilung repräsentiert werden kann, so kann die Klasse durch eine *Mischung* aus zwei Gaußverteilungen dargestellt werden – je eine für den jeweiligen Schreibstil.

Ein ähnliches Beispiel bietet die Spracherkennung, bei der dasselbe Wort aufgrund von Unterschieden in der Aussprache, im Akzent, Geschlecht, Alter, und so weiter verschiedenartig klingen kann. Wenn somit kein einzelner universaler Prototyp vorhanden ist, sollten alle diese verschiedenen Aussprachen in der Dichte vertreten sein, um statistische Korrektheit zu garantieren.

SEMIPARAMETRISCHE DICHTESCHÄTZUNG

Wir bezeichnen diesen Ansatz als *semiparametrische Dichteschätzung*, da wir immer noch von einem parametrischen Modell für jede Gruppe in der Stichprobe ausgehen. Wir befassen uns mit dem *nichtparametrischen* Ansatz in Kapitel 8, welcher dann zum Einsatz kommt, wenn die Daten keiner Strukturierung unterliegen und selbst ein gemischtes Modell nicht angewendet werden kann. In diesem Kapitel konzentrieren wir uns auf die Dichteschätzung und verweisen hinsichtlich des überwachten Lernens auf Kapitel 12.

7.2 Mischungsdichten

MISCHUNGSDICHTE

Die *Mischungsdichte* wird ausgedrückt als

$$p(\boldsymbol{x}) = \sum_{i=1}^{k} p(\boldsymbol{x}|\mathcal{G}_i) P(\mathcal{G}_i) \,, \tag{7.1}$$

MISCHUNGSKOMPONENTEN, GRUPPEN

wobei $\mathcal{G}_i$ die *Mischungskomponenten* sind. Man nennt diese auch *Gruppen* oder *Cluster*. $p(\boldsymbol{x}|\mathcal{G}_i)$ sind die *Komponentendichten* und $P(\mathcal{G}_i)$ sind die *Mischungsproportionen*. k, die Anzahl an Komponenten, ist ein Hyperparameter und sollte vorher bestimmt werden. Sind eine Stichprobe und k vorhanden, dann entspricht der Lernprozess der Schätzung der Komponentendichten und Proportionen. Wenn wir davon ausgehen, dass die Komponentendichten einem parametrischen Modell gehorchen, müssen wir lediglich deren Parameter schätzen. Wenn die Komponentendichten multivariaten Gaußverteilungen unterliegen, haben wir $p(\boldsymbol{x}|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, und $\Phi = \{P(\mathcal{G}_i), \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^k$ steht für die Parameter, die anhand der unabhängigen und identisch verteilten Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ geschätzt werden sollen.

CLUSTER
KOMPONENTENDICHTEN
MISCHUNGSPROPORTIONEN

Die parametrische Klassifikation ist ein Mischungsmodell, bei dem die Gruppen $\mathcal{G}_i$ den Klassen $\mathcal{C}_i$ entsprechen, Komponentendichten $p(\boldsymbol{x}|\mathcal{G}_i)$ mit den Klassendichten $p(\boldsymbol{x}|\mathcal{C}_i)$ zu vergleichen sind, und $P(\mathcal{G}_i)$ den a-priori-Verteilungen $P(\mathcal{C}_i)$ der Klassen entsprechen:

$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|\mathcal{C}_i)P(\mathcal{C}_i) \; .$$

In diesem *überwachten* Fall wissen wir, wie viele Gruppen es gibt und das Erlernen der Parameter ist ein trivialer Prozess, weil uns die Zuordnungen vorliegen, wir also wissen, welche Instanz zu welcher Klasse (Klassenkomponente) gehört. Wir erinnern uns aus Kapitel 5, dass bei Vorgabe der Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$, mit $r_i^t = 1$, falls $\boldsymbol{x}^t \in \mathcal{C}_i$ und andernfalls $r_i^t = 0$, die Parameter mit Hilfe der Maximum-Likelihood-Methode berechnet werden können. Wenn jede Klasse einer Gaußverteilung unterliegt, haben wir eine Mischung aus Gaußverteilungen und die Parameter werden geschätzt durch

$$\begin{aligned} \hat{P}(\mathcal{C}_i) &= \frac{\sum_t r_i^t}{N} \; , \\ \boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \; , \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t} \; . \end{aligned} \tag{7.2}$$

Der Unterschied in diesem Kapitel besteht darin, dass die Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ ist; uns liegt ein Problem des *unüberwachten Lernens* vor. Nur $\boldsymbol{x}^t$ ist gegeben, nicht aber die Zuordnungen $\boldsymbol{r}^t$, das heißt, wir wissen nicht welches $\boldsymbol{x}^t$ aus welcher Komponente entstammt. Daher sollten wir beides schätzen. Zuerst sollten wir die Zuordnungen r_i^t schätzen, also die Komponente, zu der eine gegebene Instanz gehört; und zweitens sollten nach dieser Schätzung die Parameter der Komponenten für die gegebene Menge an ihnen zugehörigen Instanzen geschätzt werden. Zuerst diskutieren wir zu diesem Zweck die k-Means-Clusteranalyse, eine sehr einfache für diesen Zweck, und zeigen später, dass sie ein Sonderfall des *Expectation-Maximization-Algorithmus* (EM) ist.

7.3 k-Means-Clusteranalyse

Gehen wir einmal von einem Bild aus, welches mit 24 Bit/Pixel gespeichert ist und bis zu 16 Mio. Farben haben kann. Angenommen, wir haben einen Farbbildschirm mit 8 Bit/Pixel, der nur 256 Farben anzeigen kann. Wir wollen daher die besten 256 Farben unter allen 16 Mio. Farben

herausfinden, und zwar so, dass das Bild unter Nutzung von lediglich diesen 256 Farben der Palette dem Originalbild so ähnlich sieht wie möglich. Hierbei handelt es sich um die *Farbquantisierung*, bei der wir von hoher auf niedrigere Auflösung abbilden. Im allgemeinen Fall besteht das Ziel darin, von einem kontinuierlichen Raum auf einen diskreten Raum abzubilden; diesen Prozess nennt man *Vektorquantisierung*.

FARBQUANTISIERUNG

VEKTORQUANTISIERUNG

Natürlich können wir immer einheitlich quantisieren, jedoch wäre dies eine Verschwendung in der Farbtabelle aufgrund der Zuweisung von Einträgen zu Farben, die im Bild nicht existieren oder weil den Farben, die besonders häufig im Bild verwendet werden, keine Sondereinträge zugewiesen werden. Wenn das Bild zum Beispiel eine Meereslandschaft darstellt, erwarten wir darin viele Schattierungen der Farbe Blau und vielleicht keine der Farbe Rot. Daher sollte die Verteilung in der Farbtabelle die Originaldichte so nah wie möglich reflektieren, indem viele Einträge in den Regionen hoher Dichte eingetragen und Regionen ohne Daten verworfen werden.

Sagen wir einmal, wir haben eine Stichprobe von $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ und k *Referenzvektoren*, $\boldsymbol{m}_j, j = 1, \ldots, k$. In unserem Beispiel der Farbquantisierung sind $\boldsymbol{x}^t$ die Bildpixelwerte in 24 Bit und $\boldsymbol{m}_j$ sind die Farbtabelleneinträge, ebenfalls in 24 Bit, mit $k = 256$.

REFERENZVEKTOR

Nehmen wir für den Moment an, dass wir auf irgendeine Weise zu den Werten von $\boldsymbol{m}_j$ gekommen sind; wir werden uns in Kürze damit befassen, wie wir sie lernen können. Bei der Bilddarstellung dann repräsentieren wir ein gegebenes Pixel $\boldsymbol{x}^t$ mit dem ähnlichsten Eintrag $\boldsymbol{m}_i$ aus der Farbtabelle und befolgen

$$\|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| .$$

Das heißt, statt der originalen Datenwerte nutzen wir den naheliegendsten Wert, den wir im Alphabet der Referenzvektoren finden. $\boldsymbol{m}_i$ nennt man auch *Codebuchvektoren* oder *Codewörter*, weil wir es hier mit einem *Codier-* und *Decodierprozess* zu tun haben (siehe Abbildung 7.1). Von $\boldsymbol{x}^t$ zu i zu gelangen ist ein Prozess, bei dem Daten mit Hilfe des Codebuchs von $\boldsymbol{m}_i, i = 1, \ldots, k$ codiert werden und bei dem auf Empfängerseite die Generierung von $\boldsymbol{m}_i$ aus i dem Decodieren entspricht. Die Quantisierung erlaubt auch eine *Kompression*. Statt beispielsweise 24 Bit zu nutzen, um jedes $\boldsymbol{x}^t$ zu speichern (oder über einen Kommunikationspfad zu transferieren), können wir einfach den zugehörigen Index i in der Farbtabelle speichern/übertragen und dabei 8 Bit nutzen, um jeden der 256 Werte zu indizieren; somit erhalten wir eine Kompressionsrate von fast 3; die Farbtabelle muss ja schließlich auch noch gespeichert/übertragen werden.

CODEBUCHVEKTOR
CODEWORT

KOMPRESSION

Befassen wir uns nun damit, wie wir $\boldsymbol{m}_i$ berechnen können. Wenn $\boldsymbol{x}^t$ durch $\boldsymbol{m}_i$ repräsentiert wird, dann gibt es einen Fehler der proportional zur Entfernung $\|\boldsymbol{x}^t - \boldsymbol{m}_i\|$ ist. Damit das neue Bild genauso aussieht wie das Originalbild, sollten diese Distanzen für alle Pixel so klein wie

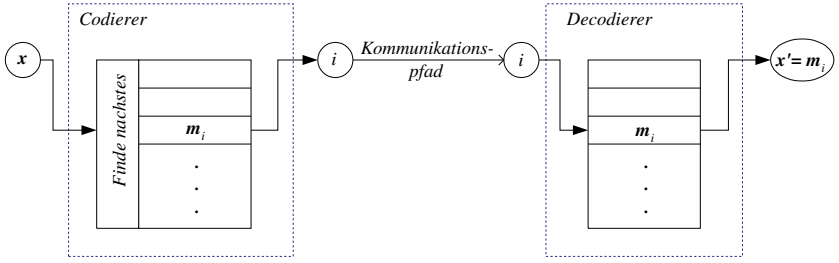


Abb. 7.1: Bei gegebenem $\boldsymbol{x}$ sendet der Codierer den Index des naheliegendsten Codewortes und der Decodierer generiert das Codewort mit dem empfangenen Index als $\boldsymbol{x}'$. Der Fehler ist $\|\boldsymbol{x}' - \boldsymbol{x}\|^2$.

möglich sein. Der gesamte *Rekonstruktionsfehler* definiert sich als

Rekonstruktions-
fehler

$$E(\{\boldsymbol{m}_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 \tag{7.3}$$

mit

$$b_i^t = \begin{cases} 1 \text{ falls } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \,, \\ 0 \text{ andernfalls} \,. \end{cases} \tag{7.4}$$

Die besten Referenzvektoren sind diejenigen, die den gesamten Rekonstruktionsfehler minimieren. b_i^t hängen auch von $\boldsymbol{m}_i$ ab, und wir können dieses Optimierungsproblem nicht analytisch lösen. Wir haben dafür eine iterative Prozedur zur Verfügung, die *k-Means-Clusteranalyse*. Als erstes beginnen wir mit einigen zufällig initiierten $\boldsymbol{m}_i$. Bei jeder Iteration nutzen wir dann zuerst Gleichung 7.4 und berechnen b_i^t für alle $\boldsymbol{x}^t$, was den *geschätzten Zuordnungen* entspricht; wenn b_i^t gleich 1 ist, so sagen wir, dass $\boldsymbol{x}^t$ zur Gruppe von $\boldsymbol{m}_i$ gehört. Wenn wir dann diese Zuordnung einmal haben, minimieren wir Gleichung 7.3. Indem wir bezüglich $\boldsymbol{m}_i$ ableiten und das Ganze gleich 0 setzen, erhalten wir

k-Means-
Clusteranalyse

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t} \,. \tag{7.5}$$

Der Referenzvektor ist gleichgesetzt mit dem Mittelwert aller Instanzen, die er repräsentiert. Man beachte, dass es sich hierbei um die gleiche Formel handelt wie die für den Mittelwert in Gleichung 7.2, außer dass wir die geschätzten Zuordnungen b_i^t anstelle der Zuordnungen r_i^t einsetzen. Es handelt sich außerdem um eine iterative Prozedur, da sich nach der Berechnung von $\boldsymbol{m}_i$ die b_i^t ändern und neuberechnet werden müssen, was wiederum $\boldsymbol{m}_i$ beeinflusst. Diese zwei Schritte werden solange wiederholt, bis die $\boldsymbol{m}_i$ sich stabilisieren (siehe Abbildung 7.2). Der Pseudocode des k-Means-Algorithmus ist im Listing 7.1 gegeben.

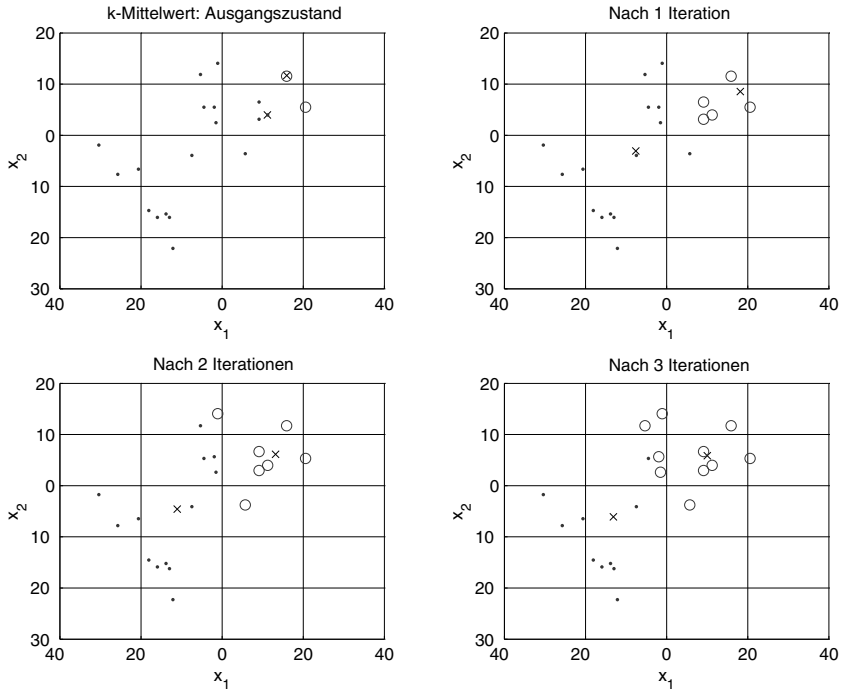


Abb. 7.2: Evolution des k-Means-Algorithmus. Kreuze zeigen die Positionen von Zentren an. Datenpunkte sind in Abhängigkeit des naheliegendsten Zentrums markiert.

Initialisiere $\boldsymbol{m}_i, i = 1, \ldots, k$, zum Beispiel mit k zufälligen $\boldsymbol{x}^t$
Repeat
  For alle $\boldsymbol{x}^t \in \mathcal{X}$
    $b_i^t \leftarrow \begin{cases} 1 \text{ falls } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 \text{ andernfalls} \end{cases}$
  For alle $\boldsymbol{m}_i, i = 1, \ldots, k$
    $\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$
Until $\boldsymbol{m}_i$ konvergieren

Listing 7.1: k-Means-Algorithmus.

Ein Nachteil liegt darin, dass es sich um eine lokale Suchprozedur handelt und dass die finalen $\boldsymbol{m}_i$ in großem Maße von den anfänglichen $\boldsymbol{m}_i$ abhängen. Es gibt diverse Methoden zur Initialisierung:

- Wir können einfach zufällig ausgewählte k Instanzen als die anfänglichen $\boldsymbol{m}_i$ nutzen.

- Der Mittelwert aller Daten kann berechnet werden und kleine zufällige Vektoren können zum Mittelwert hinzugefügt werden, um die k anfänglichen $\boldsymbol{m}_i$ zu erhalten.
- Wir können die Hauptkomponente berechnen, ihren Wertebereich in k gleiche Intervalle teilen, wodurch die Daten in k Gruppen aufgeteilt werden, und dann die Mittelwerte dieser Gruppen als die anfänglichen Zentren nutzen.

Nach der Konvergenz sollten alle Zentren eine Teilmenge der Dateninstanzen abdecken und nützlich sein; daher ist es am besten, Zentren zu initialisieren, von denen wir glauben, dass sich dort Daten befinden.

Es gibt auch Algorithmen für das *inkrementelle* Hinzufügen von Zentren oder das Entfernen von Zentren, denen keine Dateninstanzen zugeordnet werden. Beim *Leader-Cluster-Algorithmus* verursacht eine Instanz, welche weit weg von existierenden Zentren liegt (definiert durch einen Schwellwert), die Erstellung eines neuen Zentrums an jenem Punkt (wir diskutieren solch einen Algorithmus für neuronale Netze, ART, in Kapitel 12). Auch kann ein Zentrum, dem eine große Anzahl an Instanzen ($\sum_t b_i^t / N > \theta$) zugeordnet wurde, dupliziert werden (der Kopie wird dabei ein kleiner zufälliger Vektor hinzugefügt, damit sie sich unterscheiden). Auf ähnliche Weise kann ein Zentrum, welches zu wenige Instanzen abdeckt, entfernt und ausgehend von einem anderen Teil des Eingaberaums neu gestartet werden.

Leader-Cluster-Algorithmus

Der k-Means-Algorithmus wird bei der Clusteranalyse verwendet, das heißt, beim Auffinden von Gruppierungen in den Daten, wobei die Gruppierungen durch ihre Zentren repräsentiert werden, welche wiederum die typischen Vertreter der Gruppen sind. Die Vektorquantisierung stellt eine mögliche Anwendung der Clusteranalyse dar, jedoch kann sie ebenfalls für die *Vorbearbeitung* vor einem späteren Arbeitsschritt einer Klassifikation oder Regression verwendet werden. Wenn wir b_i^t bei gegebenem $\boldsymbol{x}^t$ berechnen, vollführen wir eine Abbildung vom Originalraum auf den k-dimensionalen Raum, das heißt, auf eine der Ecken des k-dimensionalen Hyperwürfels. Eine Regressions- oder Diskriminanzfunktion kann dann in diesem neuen Raum gelernt werden; wir diskutieren solche Methoden in Kapitel 12.

7.4 Expectation-Maximization-Algorithmus

Beim k-Means näherten wir uns der Clusteranalyse als einem Problem, bei dem bestimmte Codebuchvektoren gefunden werden sollen, welche den gesamten Rekonstruktionsfehler minimieren. In diesem Abschnitt ist unser Ansatz nun probabilistisch und wir suchen die Parameter der

Komponentendichte, welche die Likelihood der Stichprobe maximieren. Unter Nutzung des gemischten Modells aus Gleichung 7.1 ergibt sich die Log-Likelihood der Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ als

$$\begin{aligned}\mathcal{L}(\Phi|\mathcal{X}) &= \log \prod_t p(\boldsymbol{x}^t|\Phi) \\ &= \sum_t \log \sum_{i=1}^{k} p(\boldsymbol{x}^t|\mathcal{G}_i) P(\mathcal{G}_i) \,, \end{aligned} \tag{7.6}$$

wobei Φ sowohl die a-priori-Wahrscheinlichkeiten $P(\mathcal{G}_i)$ als auch die ausreichenden Statistiken der Komponentendichten $p(\boldsymbol{x}^t|\mathcal{G}_i)$ enthält. Unglücklicherweise können wir das Ganze nicht analytisch für die Parameter lösen und müssen auf eine iterative Optimierung zurückgreifen.

EM-Algorithmus

Der *Expectation-Maximization-Algorithmus* (kurz: EM-Algorithmus) (Dempster, Laird und Rubin 1977; Redner und Walker 1984) wird bei der Maximum-Likelihood-Schätzung verwendet, wenn das Problem zwei Mengen von Variablen umfasst, von denen eine, X, beobachtbar und die andere, Z, verborgen ist. Das Ziel des Algorithmus besteht darin, den Parametervektor Φ zu finden, der die Likelihood der beobachteten Werte von X maximiert, $\mathcal{L}(\Phi|\mathcal{X})$. In Fällen jedoch, in denen dies nicht machbar ist, assoziieren wir die zusätzlichen *verborgenen Variablen* Z und nutzen beide Variablenmengen, um das zugrundeliegende Modell auszudrücken, damit die Likelihood der gemeinsamen Verteilung von X und Z, die *komplette* Likelihood $\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z})$, maximiert wird.

Da die Z Werte nicht beobachtet werden, können wir nicht direkt mit der kompletten Likelihood $\mathcal{L}_c$ der Daten arbeiten; stattdessen nutzen wir den Erwartungswert $\mathcal{Q}$ bei gegebenem $\mathcal{X}$ und den gegenwärtigen Parameterwerten Φ^l, wobei l die Iteration indiziert. Darauf bezieht sich die Bezeichnung *Erwartung* (E) im Algorithmus. Beim dann folgenden *Maximierungsschritt* (M) suchen wir die neuen Parameterwerte Φ^{l+1}, welche das Ganze maximieren. Somit folgt:

$$\begin{aligned}&\text{E-Schritt} : \mathcal{Q}(\Phi|\Phi^l) = E[\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \Phi^l] \,, \\ &\text{M-Schritt} : \Phi^{l+1} = \operatorname*{argmax}_{\Phi} \mathcal{Q}(\Phi|\Phi^l) \,. \end{aligned}$$

Dempster, Laird und Rubin (1977) haben bewiesen, dass eine Erhöhung in $\mathcal{Q}$ eine Erhöhung der unvollständigen Likelihood impliziert:

$$\mathcal{L}(\Phi^{l+1}|\mathcal{X}) \geq \mathcal{L}(\Phi^l|\mathcal{X}) \,.$$

Im Fall der gemischten Modelle sind die verborgenen Variablen die Quellen für Beobachtungen, sprich, welche Observation zu welcher Komponente gehört. Wenn diese gegeben wären, beispielsweise als Klassenzuordnungen

in einer überwachten Umgebung, wüssten wir, welche Parameter geändert werden müssten, um jenen Datenpunkt anzupassen. Der EM-Algorithmus funktioniert wie folgt: Im E-Schritt schätzen wir diese Zuordnungen anhand unserer momentanen Kenntnis der Komponenten, und im M-Schritt aktualisieren wir unser Wissen über die Klassen aufgrund der im E-Schritt geschätzten Zuordnungen. Diese zwei Schritte sind dieselben wie beim k-Means, nämlich die Berechnung von b_i^t (E-Schritt) und die erneute Schätzung von $\boldsymbol{m}_i$ (M-Schritt).

Wir definieren einen Vektor an *Indikatorvariablen* $\boldsymbol{z}^t = \{z_1^t, \ldots, z_k^t\}$, wobei $z_i^t = 1$, falls $\boldsymbol{x}^t$ zum Cluster $\mathcal{G}_i$ gehört, und $z_i^t = 0$ in allen anderen Fällen. $\boldsymbol{z}$ ist eine multinomiale Verteilung aus k Kategorien mit a-priori-Wahrscheinlichkeiten π_i, kurz für $P(\mathcal{G}_i)$. Dann ist

$$P(\boldsymbol{z}^t) = \prod_{i=1}^{k} \pi_i^{z_i^t} . \tag{7.7}$$

Die Wahrscheinlichkeit des Eintretens einer Beobachtung $\boldsymbol{x}^t$ ist gleich der Wahrscheinlichkeit, die durch die Komponente, durch welche die Beobachtung generiert wurde, spezifiziert ist:

$$p(\boldsymbol{x}^t|\boldsymbol{z}^t) = \prod_{i=1}^{k} p_i(\boldsymbol{x}^t)^{z_i^t} . \tag{7.8}$$

$p_i(\boldsymbol{x}^t)$ ist die Kurzform für $p(\boldsymbol{x}^t|\mathcal{G}_i)$. Die gemeinsame Dichte ist

$$p(\boldsymbol{x}^t, \boldsymbol{z}^t) = P(\boldsymbol{z}^t)p(\boldsymbol{x}^t|\boldsymbol{z}^t)$$

und die komplette Likelihood der Daten der unabhängigen und identisch verteilten Stichprobe $\mathcal{X}$ ist

$$\begin{aligned}
\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z}) &= \log \prod_t p(\boldsymbol{x}^t, \boldsymbol{z}^t|\Phi) \\
&= \sum_t \log p(\boldsymbol{x}^t, \boldsymbol{z}^t|\Phi) \\
&= \sum_t \log P(\boldsymbol{z}^t|\Phi) + \log p(\boldsymbol{x}^t|\boldsymbol{z}^t, \Phi) \\
&= \sum_t \sum_i z_i^t [\log \pi_i + \log p_i(\boldsymbol{x}^t|\Phi)] .
\end{aligned}$$

E-Schritt: Wir definieren

$$\begin{aligned}
\mathcal{Q}(\Phi|\Phi^l) &\equiv E\left[\log P(X, Z)|\mathcal{X}, \Phi^l\right] \\
&= E\left[\mathcal{L}_c(\Phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \Phi^l)\right] \\
&= \sum_t \sum_i E[z_i^t|\mathcal{X}, \Phi^l][\log \pi_i + \log p_i(\boldsymbol{x}^t|\Phi^l)] ,
\end{aligned}$$

wobei

$$\begin{aligned} E[z_i^t|\mathcal{X},\Phi^l] &= E[z_i^t|\boldsymbol{x}^t,\Phi^l] \quad \boldsymbol{x}^t \text{ sind unabh. und ident. verteilt} \\ &= P(z_i^t=1|\boldsymbol{x}^t,\Phi^l) \quad z_i^t \text{ ist eine } 0/1\text{-Zufallsvariable} \\ &= \frac{p(\boldsymbol{x}^t|z_i^t=1,\Phi^l)P(z_i^t=1|\Phi^l)}{p(\boldsymbol{x}^t|\Phi^l)} \quad \text{Satz von Bayes} \\ &= \frac{p_i(\boldsymbol{x}^t|\Phi^l)\pi_i}{\sum_j p_j(\boldsymbol{x}^t|\Phi^l)\pi_j} \\ &= \frac{p(\boldsymbol{x}^t|\mathcal{G}_i,\Phi^l)P(\mathcal{G}_i)}{\sum_j p(\boldsymbol{x}^t|\mathcal{G}_j,\Phi^l)P(\mathcal{G}_j)} \\ &= P(\mathcal{G}_i|\boldsymbol{x}^t,\Phi^l) \equiv h_i^t\,. \end{aligned} \tag{7.9}$$

Wir erkennen, dass der Erwartungswert der verborgenen Variable, $E[z_i^t]$, der a-posteriori-Wahrscheinlichkeit dafür entspricht, dass $\boldsymbol{x}^t$ durch die Komponente $\mathcal{G}_i$ generiert wird. Da es sich um eine Wahrscheinlichkeit handelt, liegt der Wert zwischen 0 und 1 und ist eine „weiche“ Zuordnung, im Gegensatz zur „harten“ 0/1-Zuordnung des k-Means.

M-Schritt: Wir maximieren $\mathcal{Q}$, um die nächste Menge an Parameterwerten, Φ^{l+1}, zu erhalten:

$$\Phi^{l+1} = \operatorname*{argmax}_{\Phi} \mathcal{Q}(\Phi|\Phi^l)\,.$$

Dies entspricht

$$\begin{aligned} \mathcal{Q}(\Phi|\Phi^l) &= \sum_t \sum_i h_i^t[\log\pi_i + \log p_i(\boldsymbol{x}^t|\Phi^l)] \\ &= \sum_t \sum_i h_i^t \log\pi_i + \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi^l)\,. \end{aligned} \tag{7.10}$$

Der zweite Term ist unabhängig von π_i und unter Nutzung der Einschränkung, dass $\sum_i \pi_i = 1$ der Lagrange-Komponente entspricht, lösen wir auf nach

$$\nabla_{\pi_i} \sum_t \sum_i h_i^t \log\pi_i - \lambda\left(\sum_i \pi_i - 1\right) = 0$$

und erhalten

$$\pi_i = \frac{\sum_t h_i^t}{N}\,, \tag{7.11}$$

was analog ist zur Berechnung der a-priori-Verteilungen in Gleichung 7.2.

Ähnlich dazu ist der erste Term von Gleichung 7.10 unabhängig von den Komponenten und kann während der Schätzung der Komponentenparameter weggelassen werden. Wir lösen auf nach

$$\nabla_\Phi \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi) = 0\,. \tag{7.12}$$

Wenn wir von gaußverteilten Komponenten – $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S}_i)$ – ausgehen, so ist der M-Schritt

$$\begin{aligned} \boldsymbol{m}_i^{l+1} &= \frac{\sum_t h_i^t \boldsymbol{x}^t}{\sum_t h_i^t}\,, \\ \mathbf{S}_i^{l+1} &= \frac{\sum_t h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{\sum_t h_i^t}\,, \end{aligned} \tag{7.13}$$

wohingegen wir für gaußverteilte Komponenten im E-Schritt Folgendes berechnen:

$$h_i^t = \frac{\pi_i |\mathbf{S}_i|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1}(\boldsymbol{x}^t - \boldsymbol{m}_i)]}{\sum_j \pi_j |\mathbf{S}_j|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_j)^T \mathbf{S}_j^{-1}(\boldsymbol{x}^t - \boldsymbol{m}_j)]}\,. \tag{7.14}$$

Erneut ist die Ähnlichkeit zwischen Gleichung 7.13 und 7.2 kein Zufall; die geschätzten weichen Zuordnungen h_i^t ersetzen die eigentlichen (unbekannten) Zuordnungen r_i^t.

Der EM-Algorithmus wird durch den k-Means initialisiert. Nach einigen Iterationen des k-Means erhalten wir die Schätzungen für die Zentren $\boldsymbol{m}_i$, und unter Nutzung der Instanzen, die von jedem der Zentren abgedeckt werden, schätzen wir die $\mathbf{S}_i$; durch $\sum_t b_i^t/N$ erhalten wir die π_i. Von dem Punkt an lassen wir den EM-Algorithmus laufen, wie in Abbildung 7.3 dargestellt ist.

Genau wie bei der parametrischen Klassifikation (Abschnitt 5.5) können wir das Ganze bei kleinen Stichproben und großer Dimensionalität regularisieren, indem wir vereinfachende Annahmen treffen. Bei $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S})$, dem Fall einer gemeinsamen Kovarianzmatrix, reduziert sich Gleichung 7.12 auf

$$\min_{\boldsymbol{m}_i, \mathbf{S}} \sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i)\,. \tag{7.15}$$

Im Falle einer gemeinsamen diagonalen Kovarianzmatrix, also $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, s^2\mathbf{I})$, erhalten wir

$$\min_{\boldsymbol{m}_i, s} \sum_t \sum_i h_i^t \frac{\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2}{s^2}\,, \tag{7.16}$$

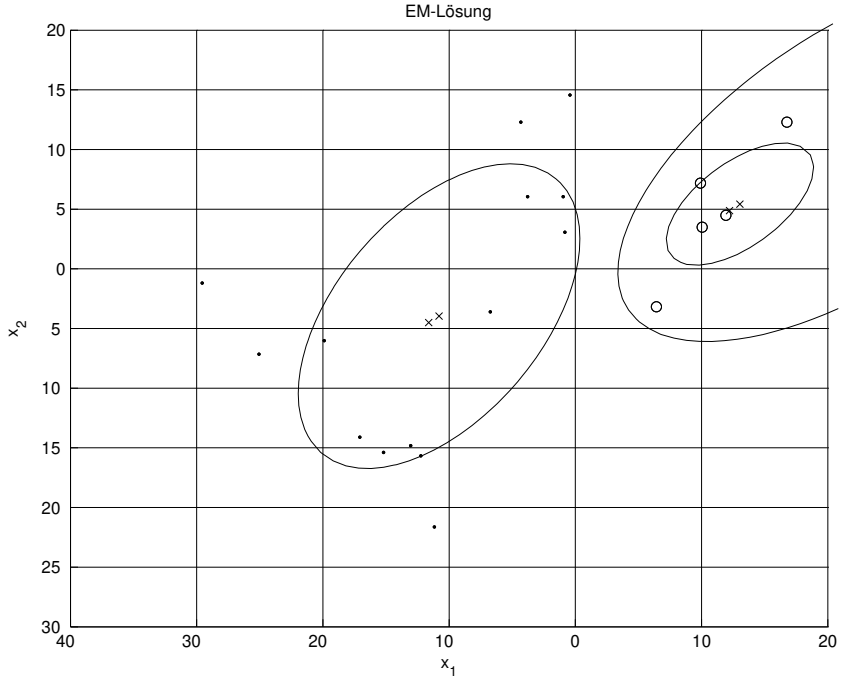


***Abb. 7.3*:** Datenpunkte und die durch den EM-Algorithmus angepassten Gaußverteilungen, initialisiert durch eine k-Means-Iteration aus Abbildung 7.2. Wie ersichtlich wird, erlaubt es der EM im Gegensatz zum k-Means, die Kovarianzmatrizen zu schätzen. Abgebildet sind die Datenpunkte, die durch größere h_i markiert sind, die Umrisse der geschätzten Gaußschen Dichten und der trennende Bogen von $h_i = 0{,}5$ (gestrichelte Linie).

was dem Rekonstruktionsfehler entspricht, den wir bei der k-Means-Clusteranalyse (Gleichung 7.3) definiert hatten. Der Unterschied besteht darin, dass jetzt

$$h_i^t = \frac{\exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2\right]}{\sum_j \exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_j\|^2\right]} \tag{7.17}$$

eine Wahrscheinlichkeit zwischen 0 und 1 ist. b_i^t bei der k-Means-Clusteranalyse trifft eine harte 0/1-Entscheidung, wohingegen h_i^t eine *weiche Zuordnung* ist, welche die Eingabe zu einem Cluster mit einer gewissen Wahrscheinlichkeit zuweist. Wenn h_i^t statt b_i^t genutzt werden, trägt eine Instanz zur Aktualisierung von Parametern aller Komponenten bei, jeweils mit einer bestimmten Wahrscheinlichkeit. Dies ist besonders nützlich, wenn die Instanz nahe der Mitte zwischen zwei Zentren liegt.

Wir sehen also, dass die k-Means-Clusteranalyse ein Sonderfall des EM-Algorithmus ist, der auf gemischte Gauß-Verteilungen angewandt wird,

bei denen Eingaben als unabhängig und mit gleicher und gemeinsamer Varianz angenommen werden, und bei denen Zuordnungen in harter Form auftreten. Bildlich gesprochen versucht also der k-Means die Eingabedichte mit Kreisen abzudecken, wohingegen der EM-Algorithmus im Allgemeinen Ellipsen nutzt, deren Form, Ausrichtung und Überdeckungsgrad beliebig sein kann.

7.5 Mischungsmodelle mit verborgenen Variablen

Wenn im Fall von Mischungen aus gaußverteilten Komponenten pro Komponente eine vollständige Kovarianzmatrix verwendet wird, besteht die Gefahr einer Überanpassung falls die Eingabedimensionalität hoch und der Stichprobenumfang klein ist, selbst wenn die Kovarianzmatrizen nicht singulär sind. Die Annahme einer gemeinsamen Kovarianzmatrix zwecks Verringerung der Anzahl an Parametern könnte sich als falsch erweisen, da Cluster wirklich unterschiedliche Formen annehmen können. Die Annahme diagonaler Matrizen ist sogar noch riskanter, da sie alle Korrelationen entfernt.

Die Alternative besteht darin, eine Dimensionalitätsreduktion in den Clustern vorzunehmen. Dies verringert die Anzahl an Parametern, während gleichzeitig die Korrelationen noch erfasst werden. Die Anzahl an freien Parametern wird durch die Dimensionalität des reduzierten Raumes kontrolliert.

Wenn wir eine Faktorenanalyse in den Clustern durchführen (siehe Abschnitt 6.5), suchen wir nach *latenten* oder *verborgenen Variablen* oder *Faktoren*, welche die Daten in den Clustern generieren (Bishop 1999):

$$p(\boldsymbol{x}^t|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{V}_i\mathbf{V}_i^T + \boldsymbol{\Psi}_i)\,, \tag{7.18}$$

wobei $\mathbf{V}_i$ und $\boldsymbol{\Psi}_i$ die Faktorenladungen und spezielle Varianzen von Cluster $\mathcal{G}_i$ sind. Rubin und Thayer (1982) stellen EM-Gleichungen für die Faktorenanalyse bereit. Es ist möglich, dies in den gemischten Modellen zu erweitern, um *Mischungen aus Faktorenanalysatoren* zu finden (Ghahramani und Hinton 1997). Im E-Schritt (in Gleichung 7.9) verwenden wir Gleichung 7.18, und im M-Schritt lösen wir Gleichung 7.12 für $\mathbf{V}_i$ und $\boldsymbol{\Psi}_i$ anstatt für $\mathbf{S}_i$. Ähnlich dazu kann man auch eine PCA in Gruppen durchführen, was man als *Mischungen von probabilistischen Hauptkomponentenanalysatoren* bezeichnet (Tipping und Bishop 1999).

GEMISCHTE FAKTORENANALYSATOREN

MISCHUNGEN AUS PROBABILISTISCHEN HAUPTKOMPONENTENANALYSATOREN

Natürlich können wir EM nutzen, um $\mathbf{S}_i$ zu lernen und dann eine FA oder PCA getrennt für jedes Cluster ausführen, jedoch ist die Anwendung von EM besser, da diese Methode die beiden Schritte – Clusteranalyse und Dimensionalitätsreduktion – miteinander koppelt und eine weiche Partitionierung vornimmt. Eine Instanz trägt zur Berechnung der latenten Variablen aller Gruppen bei, gewichtet durch h_i^t.

7.6 Überwachtes Lernen nach einer Clusteranalyse

Ebenso wie die in Kapitel 6 behandelten Verfahren zur Dimensionalitätsreduktion kann die Clusteranalyse zu zwei Zwecken verwendet werden. Erstens werden sie zur Datenexploration genutzt, um die Struktur der Daten zu verstehen. Zweitens werden sie verwendet, um Daten in einen neuen Raum abzubilden, in dem überwachtes Lernen einfacher ist.

Methoden zur Reduktion der Dimensionalität werden eingesetzt, um Korrelationen zwischen Variablen zu finden und somit Variablen zu gruppieren; die Clusteranalyse hingegen dient zum Auffinden von Ähnlichkeiten zwischen Instanzen, d. h., sie gruppiert Instanzen. Wenn solche Gruppen gefunden werden, können diese (durch Anwendungsexperten) benannt und ihre Attribute definiert werden. Man kann den Gruppenmittelwert als den repräsentativen Prototyp für Instanzen dieser Gruppe wählen oder aber auch den möglichen Wertebereich der Attribute angeben. Dies erlaubt eine einfachere Beschreibung der Daten. Wenn zum Beispiel die Kunden einer Firma in eine von k Gruppen – auch *Segmente* genannt – zu fallen scheinen, wobei Kunden hinsichtlich ihrer demographischen Attribute und ihrer Transaktionen mit der Firma beschrieben werden, dann entsteht ein besseres Verständnis für den Kundenkreis. Dadurch wird die Firma in die Lage versetzt, verschiedene Strategien für die verschiedenen Typen von Kunden zu entwickeln; dies ist Teil des *Customer-Relationship-Managements*, kurz CRM (auch *Kundenbeziehungsmanagement*). Auf dieselbe Weise erhält die Firma auch die Möglichkeit, Strategien für die Kunden zu entwickeln, die in keine größere Gruppe fallen und unter Umständen besondere Aufmerksamkeit erfordern, beispielsweise Spesenkunden.

Kundensegmentierung

Customer-Relationship-Management

Häufig wird die Clusteranalyse auch als Vorbearbeitungsstufe genutzt. Genau wie die Methoden zur Dimensionalitätsreduktion aus Kapitel 6 es uns ermöglichten, eine Abbildung auf einen neuen Raum durchzuführen, bilden wir nach der Clusteranalyse auch auf einen neuen k-dimensionalen Raum ab, dessen Dimensionen h_i sind (oder b_i auf die Gefahr hin, Informationen zu verlieren). In einer überwachten Umgebung können wir dann die Diskriminanz- oder Regressionsfunktion in diesem neuen Raum lernen. Der Unterschied zu den Methoden zur Dimensionalitätsreduktion wie der PCA besteht jedoch darin, dass k, die Dimensionalität des neuen Raumes, größer als d sein kann, also die Originaldimensionalität.

Wenn wir eine Methode wie die PCA nutzen, bei der die Dimensionen Kombinationen aus den Originaldimensionen sind, dann gehen alle Dimensionen in den Prozess ein, wenn eine beliebige Instanz im neuen Raum dargestellt werden soll, das heißt, alle z_j sind ungleich null. Im Falle einer Methode wie der Clusteranalyse, bei der neue Dimensionen lokal definiert werden, gibt es viel mehr neue Dimensionen b_j aber nur eine (oder wenn wir h_j nutzen, dann einige wenige) von ihnen hat einen

Wert ungleich null. Im ersteren Fall gibt es wenige Dimensionen, aber alle tragen etwas bei. Wir haben dann eine *verteilte Repräsentation*. Im letzteren Fall gibt es viele Dimensionen, aber nur wenige tragen etwas bei. Dann haben wir eine *lokale Repräsentation*.

LOKALE VS. VERTEILTE REPRÄSENTATION

Ein Vorteil, einem überwachten Lerner die unüberwachte Clusteranalyse oder die Dimensionalitätsreduktion voranzustellen, besteht darin, dass letztere keine zugeordneten Daten benötigen. Die Zuordnung der Daten kann kostspielig sein. Wir können eine große Menge an Daten ohne Zuordnung nutzen, um die Clusterparameter zu lernen und dann eine kleinere Datenmenge mit Zuordnungen verwenden, um die zweite Stufe der Klassifikation oder Regression zu lernen. Unüberwachtes Lernen kann man auch als „Lernen was normalerweise passiert“ beschreiben (Barrow 1989). Wenn ein überwachter Lerner folgt, lernen wir zuerst, was normalerweise passiert und dann, was das eigentlich bedeutet. Wir befassen uns mit solchen Methoden in Kapitel 12.

Wenn im Falle der Klassifikation jede Klasse ein Mischungsmodell bestehend aus einer Anzahl von Komponenten ist, so ist die gesamte Dichte eine *Mischung aus Mischungsmodellen*:

MISCHUNG AUS MISCHUNGSMODELLEN

$$p(\boldsymbol{x}|\mathcal{C}_i) = \sum_{j=1}^{k_i} p(\boldsymbol{x}|\mathcal{G}_{ij}) P(\mathcal{G}_{ij}) \,,$$
$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|\mathcal{C}_i) P(\mathcal{C}_i) \,,$$

wobei k_i die Anzahl an Komponenten ist, aus denen $p(\boldsymbol{x}|\mathcal{C}_i)$ besteht und $\mathcal{G}_{ij}$ für die Komponente j der Klasse i steht. Dabei ist zu beachten, dass unterschiedliche Klassen eventuell unterschiedlich viele Komponenten benötigen. Das Lernen der Parameter von Komponenten wird separat für jede Klasse durchgeführt (möglicherweise nach einiger Regularisierung), wie bereits besprochen wurde. Dies ist besser, als viele Komponenten auf Daten aus allen Klassen anzupassen und sie dann später mit Klassenzuordnungen zu versehen.

7.7 Spektrale Clusteranalyse

Anstatt die Clusteranalyse im Originalraum durchzuführen, kann man auch zunächst die Daten in einen neuen Raum mit reduzierter Dimensionalität abbilden, so dass die Ähnlichkeiten offensichtlicher werden, und die Clusteranalyse anschließend in diesem neuen Raum ausführen. Für diesen Schritt kann jedes Verfahren der Merkmalsselektion oder Merkmalsextraktion verwendet werden. Zu diesen Verfahren gehören die Laplaceschen Eigenmaps, die wir in Abschnitt 6.12 behandelt haben und

deren Ziel es ist, die Dateninstanzen so zu platzieren, dass gegebene paarweise Ähnlichkeiten erhalten bleiben.

Nach einem solchem Mapping werden ähnliche Punkte nahe beieinander liegen, und es ist zu erwarten, dass dies die Leistungsfähigkeit des Clusterverfahrens erhöht, beispielsweise durch Verwendung von k-Means.

SPEKTRALE CLUSTERANALYSE

Dies ist die Idee hinter der *spektralen Clusteranalyse* (von Luxburg 2007). Es gibt demnach zwei Schritte:

1. Wir definieren im Originalraum eine lokale Nachbarschaft (indem wir entweder die Anzahl der Nachbarn oder einen Schwellwert für den Abstand festlegen). Für Instanzen, die in der gleichen Nachbarschaft liegen, definieren wir dann ein Ähnlichkeitsmaß – beispielsweise unter Verwendung des Gauß-Kernels –, welches eine fallende Funktion des Abstands zwischen ihnen ist. Es sei daran erinnert, dass Instanzen, die nicht in der gleichen Nachbarschaft liegen, eine Ähnlichkeit von 0 zugeordnet wird, so dass diese an beliebigen relativen Positionen platziert werden können. Mit diesem Laplace-Operator werden die Instanzen mittels Merkmalseinbettung in dem neuen Raum platziert.
2. Auf den neuen Datenkoordinaten in diesem neuen Raum führen wir k-Means aus.

In Abschnitt 6.12 haben wir für die Matrix $\mathbf{B}$ der paarweisen Ähnlichkeiten und die Diagonalmatrix $\mathbf{D}$ mit $d_i = \sum_j B_{ij}$ den Laplace-Operator des Graphen definiert als

$$\mathbf{L} = \mathbf{D} - \mathbf{B} \,.$$

Dies ist der nicht-normierte Laplace-Operator. Es gibt zwei Möglichkeiten, ihn zu normieren. Eine davon steht in engem Zusammenhang mit dem Random Walk (Shi und Malik 2000) und die andere konstruiert eine symmetrische Matrix (Ng, Jordan und Weiss 2002). Beide können zu einer besseren Leistung bei der Clusteranalyse führen:

$$\mathbf{L}_{rw} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{B} \,,$$
$$\mathbf{L}_{sym} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{D}^{-1/2} \,.$$

Es ist immer eine gute Idee, vor der Clusteranalyse eine Dimensionalitätsreduktion unter Verwendung des Euklidischen Abstands durchzuführen, falls es redundante oder korrelierte Merkmale gibt. Die Anwendung von Laplaceschen Eigenmaps macht mehr Sinn als eine multidimensionale Skalierung oder eine Hauptkomponentenanalyse, da die letzten beiden testen, ob die paarweisen Ähnlichkeiten zwischen *allen* Paaren von Instanzen erhalten bleiben, während wir uns bei Laplaceschen Eigenmaps nur um die Erhaltung der Ähnlichkeiten zwischen benachbarten Instanzen kümmern, und zwar in einem Maße, das umgekehrt proportional zu dem

Abstand zwischen ihnen ist. Dies bewirkt, dass Instanzen, welche im Originalraum nahe beieinander – also wahrscheinlich im gleichen Cluster – liegen, auch in dem neuen Raum sehr nahe beieinander platziert werden, wodurch die Arbeit für k-Means einfacher wird; dagegen werden weiter entfernte Instanzen – die wahrscheinlich zu unterschiedlichen Clustern gehören – weit entfernt platziert. Der Graph sollte immer verbunden sein, d. h., die lokale Nachbarschaft sollte groß genug sein, um Cluster zu verbinden. Es sei daran erinnert, dass die Anzahl der Eigenvektoren mit Eigenwert 0 gleich der Anzahl der Komponenten ist und dass diese Zahl 1 sein sollte.

Die Ähnlichkeiten sind zwar lokal, doch sie pflanzen sich fort. Betrachten wir drei Instanzen a, b und c. Wir nehmen an, dass a und b in der gleichen Nachbarschaft liegen, was auch für das Paar b und c, nicht aber für a und c gelten soll. Da a und b sowie b und c jeweils nahe beieinander platziert werden, wird a auch nahe bei c liegen, und diese beiden Instanzen werden wahrscheinlich demselben Cluster zugeordnet. Betrachten wir nun die Instanzen a und d, die nicht in der gleichen Nachbarschaft liegen und zwischen denen es zu viele Zwischenknoten gibt. Diese beiden werden nicht nahe beieinander platziert, und es ist sehr unwahrscheinlich, dass sie dem gleichen Cluster zugeordnet werden.

In Abhängigkeit davon, welcher Laplace-Operator benutzt wird und wie die Größe der Nachbarschaft oder die Breite der Gauß-Kurve gewählt wird, kann es unterschiedliche Ergebnisse geben, weshalb immer mehrere Parameter getestet werden sollten (von Luxburg 2009).

7.8 Hierarchische Clusteranalyse

Wir haben uns mit der Clusteranalyse aus einer probabilistischen Perspektive im Sinne einer Anpassung eines Mischungsmodells an Daten befasst bzw. im Kontext des Findens von Codewörtern, welche den Rekonstruktionsfehler minimieren. Es gibt auch Clustermethoden, die nur Ähnlichkeiten von Instanzen nutzen, ohne jegliche Bedingungen an die Daten zu stellen. Das Ziel besteht darin, derart Gruppen zu finden, dass Instanzen in einer Gruppe ähnlicher zueinander sind als Instanzen in verschiedenen Gruppen. Dies ist der bei der *hierarchischen Clusteranalyse* verwendete Ansatz.

HIERARCHISCHE CLUSTERANALYSE

Er bedarf eines Ähnlichkeitsmaßes oder äquivalent dazu der Distanz zwischen Instanzen. Im Allgemeinen wird die Euklidische Distanz genutzt, bei der sicherzustellen ist, dass alle Attribute derselben Skalierung unterliegen. Es handelt sich um einen Sonderfall der *Minkowski-Distanz* mit $p = 2$:

$$d_m(\boldsymbol{x}^r, \boldsymbol{x}^s) = \left[\sum_{j=1}^{d} (x_j^r - x_j^s)^p \right]^{1/p} .$$

Die *City-Block-Distanz* ist einfacher zu berechnen:

$$d_{cb}(\boldsymbol{x}^r, \boldsymbol{x}^s) = \sum_{j=1}^{d} |x_j^r - x_j^s| \,.$$

AGGLOMERIERENDE CLUSTERANALYSE

Ein *agglomerierender Clusteralgorithmus* beginnt mit N Gruppen, wobei jede von ihnen anfänglich eine Instanz enthält, um dann ähnliche Gruppen zusammenzufügen und größere Gruppen zu bilden, bis es nur noch eine Gruppe gibt. Ein *divisiver Clusteralgorithmus* arbeitet in entgegengesetzter Richtung und beginnt mit einer einzelnen Gruppe, um dann größere Gruppen in kleinere zu teilen, bis jede Gruppe nur eine einzelne Instanz enthält.

DIVISIVE CLUSTERANALYSE

SINGLE-LINK-CLUSTERANALYSE

Bei jedem Durchlauf eines agglomerierenden Algorithmus wählen wir die zwei naheliegendsten Gruppen für die Verschmelzung aus. Bei der *Single-Link-Clusteranalyse* definiert sich diese Distanz als die kleinste Entfernung zwischen allen möglichen Paaren von Elementen der zwei Gruppen:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \min_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s) \,. \tag{7.19}$$

Als Beispiel betrachte man einen gewichteten, vollständig verbundenen Graphen, dessen Knoten Instanzen repräsentieren und der Kanten zwischen den Knoten mit Gewichten besitzt, die gleich den Entfernungen zwischen Instanzen sind. Dann entspricht eine Single-Link-Methode der Konstruktion des minimalen Spannbaums dieses Graphen.

COMPLETE-LINK-CLUSTERANALYSE

Bei der *Complete-Link-Clusteranalyse* wird die Distanz zwischen zwei Gruppen als die größte Entfernung zwischen allen möglichen Paaren gewählt:

$$d(\mathcal{G}_i, \mathcal{G}_j) = \max_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s) \,. \tag{7.20}$$

Dies sind die zwei am häufigsten genutzten Maßeinheiten, um die zwei naheliegendsten zu verschmelzenden Gruppen herauszufinden. Andere Möglichkeiten sind die sogenannte Average-Link-Methode, welche den Durchschnitt der Entfernungen zwischen allen Paaren nutzt, und der Zentroid-Abstand, welcher die Entfernung zwischen den Zentroiden (Mittelwerten) der zwei Gruppen misst.

DENDROGRAMM

Wenn eine agglomerierende Methode durchlaufen wird, dann wird das Ergebnis im Allgemeinen als eine hierarchische Struktur dargestellt, die unter dem Namen *Dendrogramm* bekannt ist. Hierbei handelt es sich um einen Baum, dessen Blätter Instanzen entsprechen, die in der Reihenfolge gruppiert sind, in der sie zusammengefügt wurden. Ein Beispiel ist in Abbildung 7.4 dargestellt. Der Baum kann dann auf jeder Ebene geschnitten werden, um die gewünschte Anzahl an Gruppen zu erhalten.

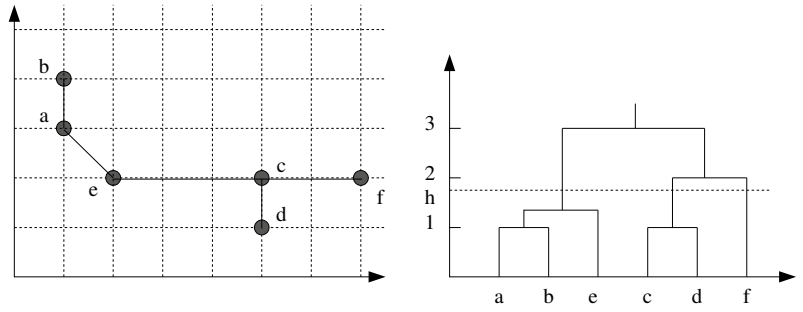


Abb. 7.4: Die Abbildung zeigt einen zweidimensionalen Datensatz und das Dendrogramm, welches das Ergebnis der Single-Link-Clusteranalyse darstellt. Man beachte, dass die Blätter des Baumes so geordnet sind, dass sich keine Äste kreuzen. Der Baum wird dann an einem gewünschten Wert von h geschnitten, um die Cluster zu erhalten.

Single-Link- und Complete-Link-Methoden berechnen die Entfernung zwischen den Gruppen unterschiedlich, welche die Cluster und das Dendrogramm beeinflussen. Bei der Single-Link-Methode werden zwei Instanzen auf Ebene h zusammengruppiert, wenn die Entfernung zwischen ihnen kleiner als h ist oder wenn es eine Übergangssequenz von Instanzen zwischen ihnen gibt, so dass die Entfernung zwischen konsekutiven Instanzen kleiner als h ist. Andererseits haben bei der Complete-Link-Methode alle Instanzen einer Gruppe eine Entfernung kleiner h zwischeneinander. Single-Link-Cluster können aufgrund dieses „Verkettungseffekts“ gestreckt werden. (Was wäre, wenn in Abbildung 7.4 eine Instanz auf halber Strecke zwischen e und c läge?) Complete-Link-Cluster tendieren dazu, kompakter zu sein.

7.9 Auswahl der Anzahl an Clustern

Wie jede Lernmethode hat auch die Clusteranalyse eine Art „Drehschalter“, um die Komplexität anzupassen; hier findet er sich in k, der Anzahl an Clustern. Bei beliebig gegebenen k wird die Clusteranalyse immer k Zentren finden, egal ob dies tatsächlich sinnvolle Gruppen sind oder ob sie nur durch die von uns benutzte Methode erzwungen werden. Es gibt verschiedene Möglichkeiten, die wir zur Feinabstimmung von k nutzen können:

- Bei einigen Anwendungen wie der Farbquantisierung ist k durch die Anwendung definiert.
- Das Abtragen der Daten in zwei Dimensionen mit Hilfe der PCA kann dafür genutzt werden, die Struktur der Daten und die Anzahl der Cluster in den Daten zu enthüllen.

- Ein inkrementeller Ansatz kann sich ebenfalls als nützlich erweisen. Die Festlegung einer maximal erlaubten Distanz ist äquivalent zur Festlegung eines maximal erlaubten Rekonstruktionsfehlers pro Instanz.
- Bei einigen Anwendungen kann die Validierung der Gruppen manuell durchgeführt werden, indem geprüft wird, ob Cluster tatsächlich sinnvolle Gruppen der Daten codieren. In Data-Mining-Anwendungen beispielsweise können Anwendungsexperten diese Überprüfung vornehmen. Bei der Farbquantisierung könnten wir das Bild visuell inspizieren, um seine Qualität zu überprüfen (trotz der Tatsache, dass unsere Augen und unser Gehirn ein Bild nicht Pixel für Pixel analysieren).

Je nachdem, welche Art von Clustermethode wir verwenden, können wir den Rekonstruktionsfehler oder die Log-Likelihood als Funktion von k graphisch darstellen und nach dem „Ellenbogen“ Ausschau halten. Nach einem ausreichend großen k beginnt der Algorithmus, Gruppen zu teilen; dann gibt es keine starke Verringerung des Rekonstruktionsfehlers und keine starke Erhöhung der Log-Likelihood. Entsprechend können wir uns bei der hierarchischen Clusteranalyse für eine gute Aufteilung entscheiden, indem wir die Unterschiede zwischen den Ebenen im Baum betrachten.

7.10 Anmerkungen

Mischungsmodelle finden häufig in der Statistik Anwendung. Lehrbücher, die sich speziell diesem Thema widmen, sind Titterington, Smith und Makov (1985) sowie McLachland und Basford (1988). McLachlan und Krishnan (1997) diskutieren neuere Entwicklungen in Sachen EM-Algorithmus, die Beschleunigung seiner Konvergenz sowie diverse Abarten. Bei der Signalverarbeitung wird der k-Means als *Linde-Buzo-Gray-Algorithmus* (LBG) bezeichnet (Gersho und Gray 1992). Er wird sowohl in der Statistik als auch bei der Signalverarbeitung häufig in einer Vielzahl von Anwendungen benutzt und tritt in vielen verschiedenen Varianten auf, wobei eine davon der sogenannte *Fuzzy-k-Means-Algorithmus* ist. Die *Fuzzy-Membership* einer Eingabe in eine Komponente ist ebenfalls ein Wert zwischen 0 und 1 (Bezdek und Pal 1995). Alpaydın (1998) vergleicht die Algorithmen k-Means, Fuzzy k-Means und den EM-Algorithmus bei gaußverteilten Mischungsmodellen. Ein Vergleich von EM und anderen Lernalgorithmen für das Erlernen von Gaußschen Mischungsmodellen findet sich bei Xu und Jordan (1996). Für kleine Datenstichproben besteht eine Alternative gegenüber der Vereinfachung von Annahmen darin, einen Bayesschen Ansatz zu nutzen (Ormoneit und Tresp 1996). Moerland (1999) stellt für eine Menge an Klassifikationsproblemen gaußverteilte Mischungen den Mischungsmodellen mit latenten Variablen gegenüber und zeigt empirisch die Vorteile der Modelle mit latenten Variablen. Ein

Fuzzy k-Means

Buch zu Clustermethoden ist das von Jain und Dubes (1988), und Übersichtsartikel zum Thema stammen von Jain, Murty und Flynn (1999) sowie von Xu und Wunsch (2005).

Einer der Vorzüge von spektralen und hierarchischen Clustermethoden besteht darin, dass wir keine Vektordarstellung der Instanzen brauchen, solange wir ein Ähnlichkeits-/Abstandsmaß zwischen den Paaren von Instanzen definieren können. Das Problem, beliebige Datenstrukturen – Dokumente, Graphen, Webseiten usw. – so durch Vektoren darzustellen, dass man einen sinnvollen Euklidischen Abstand hat, ist immer eine mühsame Aufgabe und führt zu artifiziellen Repräsentationen wie dem Bag-of-Words-Modell. Es ist daher immer gut, Maße für die (Un-)Ähnlichkeit verwenden zu können, die direkt auf der Originalstruktur definiert wurden. Den gleichen Vorteil bieten Kernel-Funktionen im Zusammenhang mit Kernel-Maschinen, die wir in Kapitel 13 behandeln.

7.11 Übungen

1. Bei der Bildkompression kann der k-Means wie folgt genutzt werden: Das Bild wird in nichtüberlappende $c \times c$ Fenster zerlegt und diese c^2-Vektoren bilden die Stichprobe. Für ein gegebenes k – normalerweise eine Potenz von 2 – führen wir eine k-Means-Clusteranalyse durch. Die Referenzvektoren und die Indices für jedes Fenster werden über den Kommunikationspfad versandt. Beim Empfänger wird das Bild dann rekonstruiert, indem mittels der Indices die Tabelle von Referenzvektoren ausgelesen wird. Schreiben Sie ein Computerprogramm, welches dies für verschiedene Werte von k und c durchführt. Berechnen Sie für jeden Fall den Rekonstruktionsfehler und die Kompressionsrate.

2. Wir können die k-Means-Clusteranalyse anwenden, die Instanzen partitionieren und dann $\mathbf{S}_i$ separat in jeder Gruppe berechnen. Warum ist das aber keine gute Idee?

   Lösung: Hierfür gibt es zwei Gründe. k-Means führt zu einer harten Partitionierung, doch es ist immer besser, eine weiche Partitionierung zu haben (also $h_i^t \in (0,1)$ anstatt $b_i^t \in \{0,1\}$ zu verwenden), so dass Instanzen (zwischen zwei Clustern) zu den Parametern (in diesem Fall ist dies die Kovarianzmatrix) von mehr als einem Cluster beitragen können, was einen glatteren Übergang zwischen den Clustern erlaubt.

   Zweitens macht k-Means Gebrauch vom Euklidischen Abstand und wir erinnern uns, dass der Euklidische Abstand Merkmale impliziert, die die gleiche Skala haben und unabhängig sind. Die Verwendung von $\mathbf{S}_i$ bedeutet, dass vom Mahalanobis-Abstand Gebrauch gemacht wird und folglich Unterschiede in den Skalen und Abhängigkeiten beachtet werden.

3. Leiten Sie die Gleichungen des M-Schritts für $\mathbf{S}$ im Falle einer gemeinsamen willkürlichen Kovarianzmatrix $\mathbf{S}$ (Gleichung 7.15) ab und für s^2 im Falle einer gemeinsamen diagonalen Kovarianzmatrix (Gleichung 7.16).

4. Definieren Sie ein multivariates gemischtes Bernoulli-Modell mit binären Eingaben und leiten Sie die EM-Gleichungen ab.

   Lösung: Wenn die Komponenten multivariate Bernoulli-Variablen sind, haben wir d-dimensionale binäre Vektoren. Angenommen, die Dimensionen sind unabhängig, dann gilt (siehe Abschnitt 5.7):

$$p_i(\boldsymbol{x}^t|\Phi) = \prod_{j=1}^{d} p_{ij}^{x_j^t}(1-p_{ij})^{1-x_j^t}$$

   mit $\Phi^l = \{p_{i1}^l, p_{i2}^l, \ldots, p_{id}^l\}_{i=1}^k$. Der E-Schritt ändert sich nicht (siehe Gleichung 7.9). Im M-Schritt maximieren wir für die Komponentenparameter $p_{ij}, i = 1, \ldots, k, j = 1, \ldots, d$

$$\begin{aligned}\mathcal{Q}' &= \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\phi^l) \\ &= \sum_t \sum_i h_i^t \sum_j x_j^t \log p_{ij}^l + (1-x_j^t)\log(1-p_{ij}^l)\,.\end{aligned}$$

   Indem wir die Ableitung nach p_{ij} bilden und diese gleich 0 setzen, erhalten wir

$$p_{ij}^{l+1} = \frac{\sum_t h_i^t x_j^t}{\sum_t h_j^t}\,.$$

   Dies ist das gleiche wie in Gleichung 5.31, mit dem Unterschied, dass die geschätzten „weichen“ Zuordnungen h_j^t die gelernten Zuordnungen r_j^t ersetzen.

5. Wie können wir bei dem Ansatz der Mischung aus Mischungen für die Klassifikation die Parameter k_i (Anzahl der Komponenten in Klasse C_i) feinabstimmen?

Editierdistanz

6. Die *Editierdistanz* zwischen zwei Zeichenketten – beispielsweise zwischen Gensequenzen – ist die Anzahl der Zeichenoperationen (Einfügen, Löschen, Ersetzen), die notwendig sind, um die eine Zeichenkette in die andere zu überführen. Welche Vorteile hat das spektrale Clusterverfahren unter Verwendung der Editierdistanz gegenüber einer gewöhnlichen k-Means-Clusteranalyse unter Verwendung der Euklidischen Distanz auf Zeichenketten?

7. Wie können wir hierarchisches Clustern mit binären Eingabevektoren – beispielsweise für das Clustern von Texten unter Verwendung der Bag-of-Words-Darstellung – erreichen?

8. Welche Gemeinsamkeiten und Unterschiede haben die Average-Link-Methode und k-Means?

   LÖSUNG: Beide messen Ähnlichkeiten anhand des Mittels der Instanzen, die in einem gemeinsamen Cluster liegen. Allerdings gibt es in einem hierarchischen Schema Cluster unterschiedlicher Auflösung.

9. Wie können wir bei der hierarchischen Clusteranalyse lokal adaptive Abstände nutzen? Was sind die Vor- und Nachteile dabei?

10. Wie können wir k-Means robust gegen Ausreißer machen?

    LÖSUNG: Ausreißer sind Instanzen, die sehr weit von *allen* Mittelpunkten entfernt liegen. Wir wollen nicht, dass diese unsere Lösung beeinflussen. Eine Möglichkeit besteht darin, sie bei der Berechnung der Parameter, beispielsweise der Mittelwerte und Kovarianzen, nicht mit zu berücksichtigen. Es ist zu beachten, dass wir die Erkennung von Ausreißern den Mahalanobis-Abstand oder die Likelihood verwenden können, nicht aber die a-posteriori-Wahrscheinlichkeit. In Abschnitt 8.7 werden wir eine nichtparametrische Methode für die Erkennung von Ausreißern behandeln.

11. Können wir ein Dendogramm, das wir generiert haben, „stutzen“?

7.12 Literaturangaben

Alpaydın, E. 1998. „Soft Vector Quantization and the EM Algorithm.“ *Neural Networks* 11: 467–477.

Barrow, H. B. 1989. „Unsupervised Learning.“ *Neural Computation* 1: 295-311.

Bezdek, J. C., and N. R. Pal. 1995. „Two Soft Relatives of Learning Vector Quantization.“ *Neural Networks* 8: 729–743.

Bishop, C. M. 1999. „Latent Variable Models,“ In *Learning in Graphical Models,* ed. M. I. Jordan. 371–403. Cambridge, MA: The MIT Press.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. „Maximum Likelihood from Incomplete Data via the EM Algorithm.“ *Journal of Royal Statistical Society* B 39: 1–38.

Gersho, A., and R. M. Gray. 1992. *Vector Quantization and Signal Compression.* Boston: Kluwer.

Ghahramani, Z., and G. E. Hinton. 1997. *The EM Algorithm for Mixtures of Factor Analyzers.* Technical Report CRG TR-96-1, Department of Computer Science, University of Toronto.

Jain, A. K., and R. C. Dubes. 1988. *Algorithms for Clustering Data.* New York: Prentice Hall.

Jain, A. K., M. N. Murty, and P. J. Flynn. 1999. „Data Clustering: A Review." *ACM Computing Surveys* 31: 264–323.

McLachlan, G. J., and K. E. Basford. 1988. *Mixture Models: Inference and Applications to Clustering.* New York: Marcel Dekker.

McLachlan, G. J., and T. Krishnan. 1997. *The EM Algorithm and Extensions.* New York: Wiley.

Moerland, P. 1999. „A Comparison of Mixture Models for Density Estimation," In *International Conference on Artificial Neural Networks,* 25–30.

Ng, A., M. I. Jordan, and Y. Weiss. 2002. „On Spectral Clustering: Analysis and an Algorithm." In *Advances in Neural Information Processing Systems 14,* ed. T. Dietterich, S. Becker, and Z. Ghahramani, 849–856. Cambridge, MA: MIT Press.

Ormoneit, D., and V. Tresp. 1996. „Improved Gaussian Mixture Density Estimates using Bayesian Penalty Terms and Network Averaging." In *Advances in Neural Information Processing Systems 8,* ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 542–548. Cambridge, MA: The MIT Press.

Redner, R. A., and H. F. Walker. 1984. „Mixture Densities, Maximum Likelihood and the EM Algorithm." *SIAM Review* 26: 195–239.

Rubin, D. B., and D. T. Thayer. 1982. „EM Algorithms for ML Factor Analysis." *Psychometrika* 47: 69–76.

Shi, J., and J. Malik. 2000. „Normalized Cuts and Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22: 888–905.

Tipping, M. E., and C. M. Bishop. 1999. „Mixtures of Probabilistic Principal Component Analyzers." *Neural Computation* 11: 443–482.

Titterington, D. M., A. F. M. Smith, and E. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions.* New York: Wiley.

von Luxburg, U. 2007. „A Tutorial on Spectral Clustering." *Statistical Computing* 17: 395–416.

Xu, L., and M. I. Jordan. 1996. „On Convergence Properties of the EM Algorithm for Gaussian Mixtures." *Neural Computation* 8: 129–151.

Xu, R., and D. Wunsch II. 2005. „Survey of Clustering Algorithms." *IEEE Transactions on Neural Networks* 16: 645–678.

8 Nichtparametrische Methoden

In den vorangegangen Kapiteln haben wir die parametrischen und semiparametrischen Ansätze betrachtet, bei denen wir annahmen, dass die Daten aus einer Wahrscheinlichkeitsverteilung oder einer Mischung aus Verteilungen bekannter Form entnommen wurden. Jetzt werden wir den nichtparametrischen Ansatz diskutieren, welcher dann genutzt wird, wenn keine solche Annahme hinsichtlich der Eingabedichte getroffen werden kann und die Daten für sich selbst sprechen. Wir behandeln die nichtparametrischen Ansätze für die Dichteschätzung, Klassifikation, Ausreißererkennung und Regression, und untersuchen, wie die Komplexität in Zeit und Raum überprüft werden kann.

8.1 Einführung

Bei den parametrischen Methoden – egal ob für Dichteschätzungen, Klassifikationen oder Regressionen – gehen wir von einem Modell aus, welches über den gesamten Eingaberaum gültig ist. Wenn wir beispielsweise bei der Regression von einem linearen Modell ausgehen, nehmen wir an, dass die Ausgabe für eine beliebige Eingabe dieselbe lineare Funktion der Eingabe ist. Setzen wir bei der Klassifikation eine Normalverteilung voraus, dann nehmen wir damit an, dass alle Beispiele der Klasse aus dieser selben Dichte gezogen wurden. Der Vorteil einer parametrischen Methode liegt darin, dass sie das Problem der Berechnung einer Wahrscheinlichkeitsdichtefunktion, Diskriminante oder Regressionsfunktion auf die Berechnung der Werte einer geringen Anzahl von Parametern reduziert. Ihr Nachteil ergibt sich daraus, dass diese Annahme nicht in jedem Fall haltbar ist und wir große Fehler verursachen, wenn eben solch ein Fall eintritt. Wenn wir derlei Annahmen nicht treffen und kein parametrisches Modell erstellen können, besteht eine Möglichkeit darin, ein semiparametrisches Mischungsmodell zu nutzen, wie in Kapitel 7 gesehen, bei dem die Dichte als Disjunktion einer kleinen Anzahl von parametrischen Modellen formuliert wird.

NICHTPARAMETRISCHE SCHÄTZUNG

Bei der *nichtparametrischen Schätzung* gehen wir lediglich davon aus, dass *ähnliche Eingaben auch ähnliche Ausgaben haben*. Hierbei handelt es sich um eine nachvollziehbare Annahme. Die Welt ist gleichmäßig gestaltet und Funktionen – egal ob es sich um Dichten, Diskriminanten oder Regressionsfunktionen handelt – ändern sich nur langsam. Ähnliche Instanzen repräsentieren ähnliche Dinge. Wir alle lieben unsere Nachbarn, weil sie uns so ähnlich sind.

Somit besteht unser Algorithmus darin, unter Verwendung eines geeigneten Entfernungsmaßes ähnliche Instanzen aus dem Datensatz zu finden und mit deren Hilfe zu interpolieren, um die richtige Ausgabe zu bestimmmen. Verschiedene nichtparametrische Methoden unterscheiden sich in der Art und Weise, wie sie Ähnlichkeit definieren oder wie sie mit Hilfe der ähnlichen Instanzen interpolieren. Bei einem parametrischen Modell beeinflussen alle Instanzen die endgültige globale Schätzung, wohingegen es beim nichtparametrischen Fall kein einzelnes globales Modell gibt; lokale Modelle werden je nach Bedarf geschätzt und nur durch die nahe liegenden Instanzen beeinflusst.

Nichtparametrische Verfahren setzen keine a-priori-Annahmen in Form von paramatrisierten Dichten voraus. Weniger streng interpretiert ist ein nichtparametrisches Modell nicht starr festgelegt, sondern seine Komplexität hängt von der Größe der Trainingsmenge ab, oder anders formuliert, von der Komplexität des Problems, die sich in den Daten niederschlägt.

INSTANZBASIERTES/ SPEICHERBASIERTES LERNEN

In der Literatur zum maschinellen Lernen spricht man bei nichtparametrischen Methoden auch von *instanzbasierten* oder *speicherbasierten* Lernalgorithmen, denn was sie tun, ist nichts anderes, als die Instanzen in einer Look-Up-Tabelle abzulegen und anhand dieser zu interpolieren. Dies impliziert, dass alle Instanzen gespeichert werden sollten, wofür sich Speicheranforderungen von $\mathcal{O}(N)$ ergeben. Weiterhin sollten bei gegebener Eingabe ähnliche Instanzen gefunden werden, und dieses Auffinden wiederum erfordert einen Rechenaufwand von $\mathcal{O}(N)$. Solche Methoden nennt man auch *faule* (lazy) Lernalgorithmen, denn im Gegensatz zu den *eifrigen* (eager) parametrischen Modellen berechnen sie kein Modell, wenn ihnen ein Datensatz vorliegt, sondern verschieben die Berechnung des Modells so lange, bis ihnen eine Testinstanz vorgesetzt wird. Im Falle eines parametrischen Ansatzes ist das Modell recht einfach und besitzt eine geringe Anzahl an Parametern, geordnet nach $\mathcal{O}(d)$ oder $\mathcal{O}(d^2)$; sobald diese Parameter einmal anhand des Datensatzes berechnet sind, behalten wir das Modell und benötigen nicht länger den Datensatz für die Kalkulation der Ausgabe. N ist im Allgemeinen viel größer als d (oder d^2), und dieser erhöhte Bedarf an Speicherplatz und Rechenaufwand bildet den Nachteil nichtparametrischer Methoden.

Wir beginnen mit der Schätzung einer Dichtefunktion und diskutieren ihre Nutzung bei der Klassifikation. Dann verallgemeinern wir den Ansatz für die Regression.

8.2 Nichtparametrische Dichteschätzung

Wie bei der Dichteschätzung üblich nehmen wir an, dass die Stichprobe $\mathcal{X} = \{x^t\}_{t=1}^N$ unabhängig aus einer unbekannten Wahrscheinlichkeitsdichte $p(\cdot)$ gezogen wurde. $\hat{p}(\cdot)$ ist unser Schätzer von $p(\cdot)$. Wir beginnen mit dem univariaten Fall, bei dem x^t Skalare sind, und verallgemeinern dann später auf den multidimensionalen Fall.

Der nichtparametrische Schätzer für die kumulative Verteilungsfunktion $F(x)$ am Punkt x ist das Verhältnis der Stichprobenpunkte, die kleiner oder gleich x sind:

$$\hat{F}(x) = \frac{\#\{x^t \leq x\}}{N} , \tag{8.1}$$

wobei $\#\{x^t \leq x\}$ für die Anzahl an Instanzen steht, deren x^t kleiner oder gleich x ist. Ähnlich dazu kann die nichtparametrische Schätzung für die Dichtefunktion berechnet werden als

$$\hat{p}(x) = \frac{1}{h}\left[\frac{\#\{x^t \leq x+h\} - \#\{x^t \leq x\}}{N}\right] . \tag{8.2}$$

h ist die Länge des Intervalls, und die Instanzen x^t, die in dieses Intervall fallen, werden als „nah genug“ angenommen. Die in diesem Kapitel vorgestellten Techniken sind Varianten, bei denen unterschiedliche Heuristiken zum Einsatz kommen, um die nahe liegenden Instanzen und ihre Auswirkungen auf die Schätzung zu bestimmen.

8.2.1 Histogrammschätzer

Die älteste und am meisten genutzte Methode ist das *Histogramm*, bei dem der Eingaberaum in Intervalle gleicher Größe (*bins*) zerlegt wird. Bei gegebenem Ursprung x_o und einer Intervallbreite h sind die Intervalle $[x_o + mh, x_o + (m+1)h)$ für positive und negative ganzzahlige m, und die Schätzung ist gegeben als

HISTOGRAMM

$$\hat{p}(x) = \frac{\#\{x^t \text{ im selben Intervall wie } x\}}{Nh} . \tag{8.3}$$

Bei der Konstruktion des Histogramms müssen wir sowohl einen Ursprung als auch eine Intervallbreite (bins) wählen. Die Wahl des Ursprungs beeinflusst die Schätzung nahe der Grenzen der Intervalle, jedoch ist es hauptsächlich die Intervallbreite, die Einfluss auf die Schätzung nimmt: bei schmalen Intervallen ist die Schätzung mit spitzen Zacken versehen, bei breiten Intervallen ist sie gleichmäßiger (siehe Abbildung 8.1). Die Schätzung ist 0, wenn keine Instanz in ein Intervall fällt und es Diskontinuitäten an den Intervallgrenzen gibt. Dennoch ist ein Vorteil des

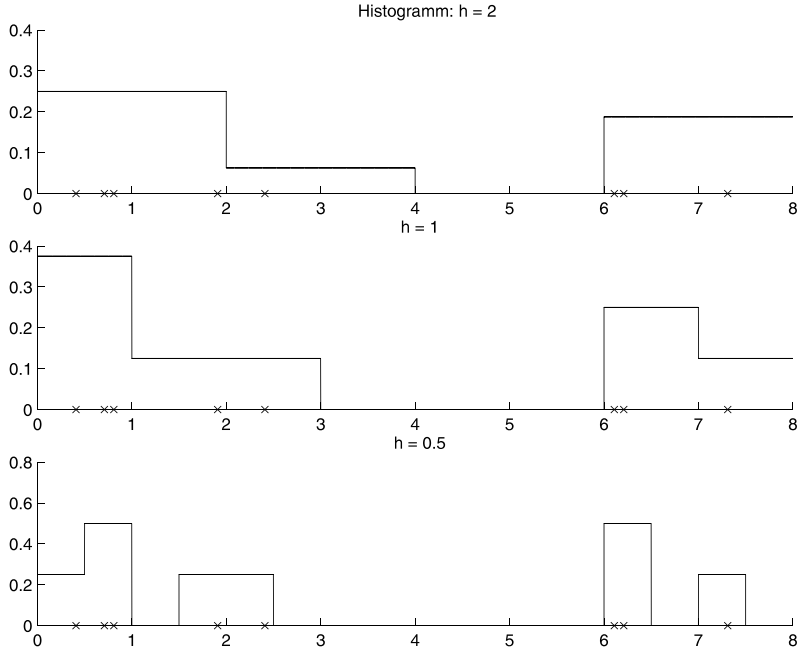


***Abb. 8.1*:** Histogramme für diverse Intervallbreiten. Die „×“ stehen für Datenpunkte.

Histogramms der, dass wir, sobald die Intervallschätzungen einmal berechnet und gespeichert wurden, den Datensatz nicht mehr zurückbehalten müssen.

NAIVER SCHÄTZER Der *naive Schätzer* (Silverman 1986) befreit uns von der Aufgabe, einen Ursprung festlegen zu müssen. Er definiert sich als

$$\hat{p}(x) = \frac{\#\{x - h < x^t \leq x + h\}}{2Nh} \tag{8.4}$$

und ist gleich der Histogrammschätzung, bei der x immer im Zentrum eines Intervalls der Größe $2h$ liegt (siehe Abbildung 8.2). Der Schätzer kann auch geschrieben werden als

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) , \tag{8.5}$$

wobei sich die *Gewichtsfunktion* definiert als

$$w(u) = \begin{cases} 1 \text{ falls } |u| < \frac{1}{2} , \\ 0 \text{ andernfalls} . \end{cases}$$

Es ist, als ob jedes x^t eine symmetrische Einflussregion von Größe h um sich herum hätte und 1 für jedes x beitragen würde, das in seine

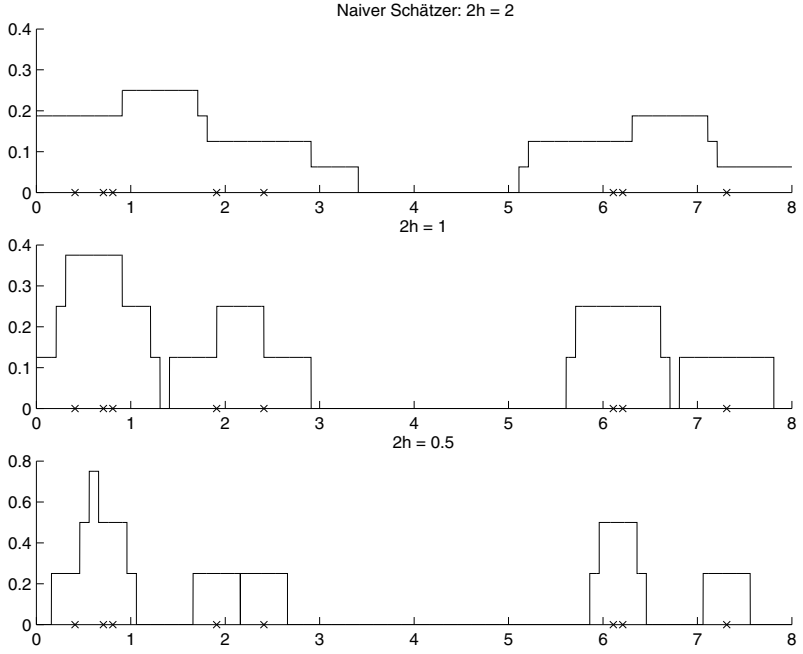


***Abb. 8.2*:** Naiver Schätzer für diverse Intervallbreiten.

Region fällt. Dann ist die nichtparametrische Schätzung lediglich die Summe der Einflüsse von allen x^t, deren Regionen x beinhalten. Da diese Einflussregion „hart“ ist (0 oder 1), ist die Schätzung keine kontinuierliche Funktion und besitzt Sprünge bei $x^t \pm h/2$.

8.2.2 Kernel-Schätzer

Um eine glatte Schätzung zu erhalten, nutzen wir eine glatte Gewichtsfunktion, die als *Kernel-Funktion* bezeichnet wird. Die am meisten verwendete ist der Gauß-Kernel: KERNEL-FUNKTION

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] . \tag{8.6}$$

Der *Kernel-Schätzer*, auch *Parzen-Fenster* genannt, ist definiert als KERNEL-SCHÄTZER PARZEN-FENSTER

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) . \tag{8.7}$$

Die Kernel-Funktion $K(\cdot)$ bestimmt die Form der Einflüsse, und die Fensterbreite h legt die Breite fest. Genauso wie die naive Schätzung

die Summe der „Boxen“ ist, ist die Kernel-Schätzung die Summe der „Schwellen“. Alle x^t bewirken einen Effekt auf die Schätzung bei x, der mit Zunahme von $|x - x^t|$ gleichmäßig abnimmt.

Um die Berechnung zu vereinfachen, kann $K(\cdot)$ als 0 gewählt werden, wenn $|x - x^t| > 3h$. Es existieren andere, einfacher zu berechnende Kernel, die genutzt werden können, sofern $K(u)$ bei $u = 0$ maximal ist und symmetrisch abnimmt, wenn $|u|$ zunimmt.

Wenn h klein ist, hat jede Instanz einen großen Effekt in einer kleinen Region und keinen Effekt auf entfernte Punkte. Ist h größer, so gibt es mehr Überlappungen der Kernel und wir erhalten eine glatte Schätzung (siehe Abbildung 8.3). Falls $K(\cdot)$ überall nicht negativ ist und zu 1 integriert, sprich, wenn es eine legitime Dichtefunktion ist, so trifft dies auch auf $\hat{p}(\cdot)$ zu. Weiterhin wird $\hat{p}(\cdot)$ die gesamte Kontinuität und die Eigenschaften der Differenzierbarkeit des Kernels $K(\cdot)$ erben, so dass, wenn $K(\cdot)$ beispielsweise eine Gaußverteilung ist, $\hat{p}(\cdot)$ glatt und beliebig differenzierbar ist.

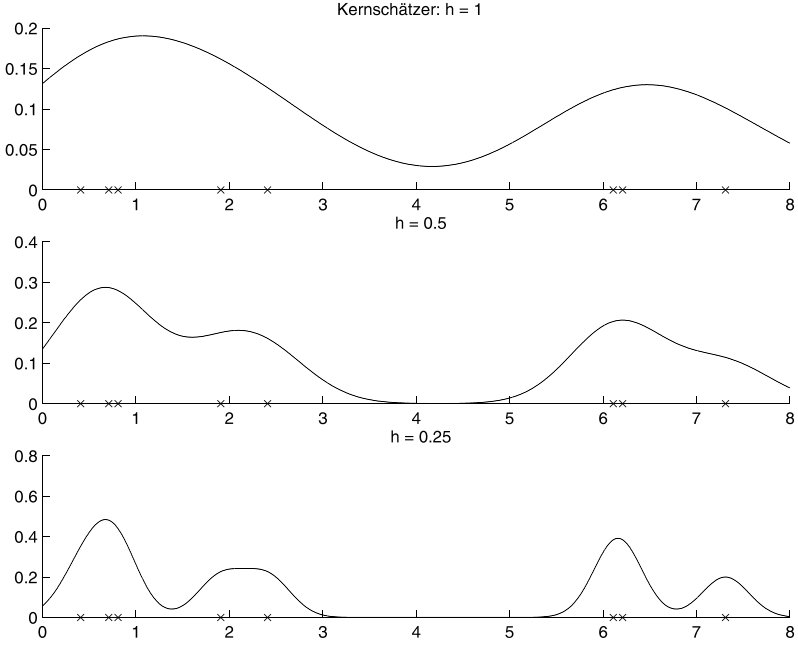


***Abb. 8.3*:** Kernel-Schätzung für diverse Intervallbreiten.

Ein Problem besteht darin, dass die Fensterbreite über den gesamten Eingaberaum festgelegt ist. Diverse adaptive Methoden wurden bereits vorgeschlagen, um h als eine Funktion der Dichte um x herum darzustellen.

8.2.3 k-Nächste-Nachbarn-Schätzer

Die „Nächste-Nachbarn"-Schätzer passen das Ausmaß der Glättung an die *lokale* Dichte der Daten an. Der Glättungsgrad wird durch k kontrolliert, also die Anzahl an betrachteten Nachbarn, die viel kleiner ist als die Stichprobengröße N. Definieren wir eine Distanz zwischen a und b, beispielsweise $|a-b|$, und legen wir für jedes x fest, dass

$$d_1(x) \leq d_2(x) \leq \cdots \leq d_N(x)$$

die Entfernungen – in aufsteigender Ordnung sortiert – von x zu den Punkten in der Stichprobe: $d_1(x)$ ist die Distanz zur nächstliegenden Stichprobe, $d_2(x)$ ist die Distanz zur übernächsten, und so weiter. Wenn x^t die Datenpunkte sind, so definieren wir $d_1(x) = \min_t |x-x^t|$, und wenn i der Index der nächstgelegenen Stichprobe ist, also $i = \operatorname{argmin}_t |x - x^t|$, dann ist $d_2(x) = \min_{j \neq i} |x - x^j|$, und so fort.

Die Dichteschätzung der *k-Nächste-Nachbarn-Methode* (auch *k-nearest neighbor*; k-NN) ist

k-Nächste-Nachbarn-Schätzung

$$\hat{p}(x) = \frac{k}{2Nd_k(x)} \,. \tag{8.8}$$

Dies entspricht einem naiven Schätzer mit $h = 2d_k(x)$, mit dem Unterschied, dass wir nicht mehr h auf einen festen Wert setzen und überprüfen, wie viele Stichproben in das Intervall fallen. Stattdessen setzen wir nun k auf einen festen Wert, also die Anzahl an Beobachtungen, die in dem Intervall liegen, und berechnen dann die Intervallbreite. Wo die Dichte hoch ist, sind die Intervalle kleiner, und wo die Dichte niedrig ist, sind sie größer (siehe Abbildung 8.4).

Der k-NN-Schätzer ist nicht kontinuierlich; seine Ableitung weist eine Diskontinuität bei allen $\frac{1}{2}(x^{(j)} + x^{(j+k)})$ auf, wobei $x^{(j)}$ der zu $d_j(x)$ zugehörige Punkt der Stichprobe ist. Die k-NN Dichteschätzung ist keine Wahrscheinlichkeitsdichtefunktion, da sie zu ∞ integriert, und nicht zu 1.

Um eine glatte Schätzung zu erhalten, können wir eine Kernel-Funktion nutzen, deren Effekt sich mit wachsender Distanz verringert:

$$\hat{p}(x) = \frac{1}{Nd_k(x)} \sum_{t=1}^{N} K\left(\frac{x - x^t}{d_k(x)}\right) \,. \tag{8.9}$$

Dies entspricht einem Kernel-Schätzer mit adaptivem Glättungsparameter $h = d_k(x)$. $K(\cdot)$ wird typischerweise als Gauß-Kernel gewählt.

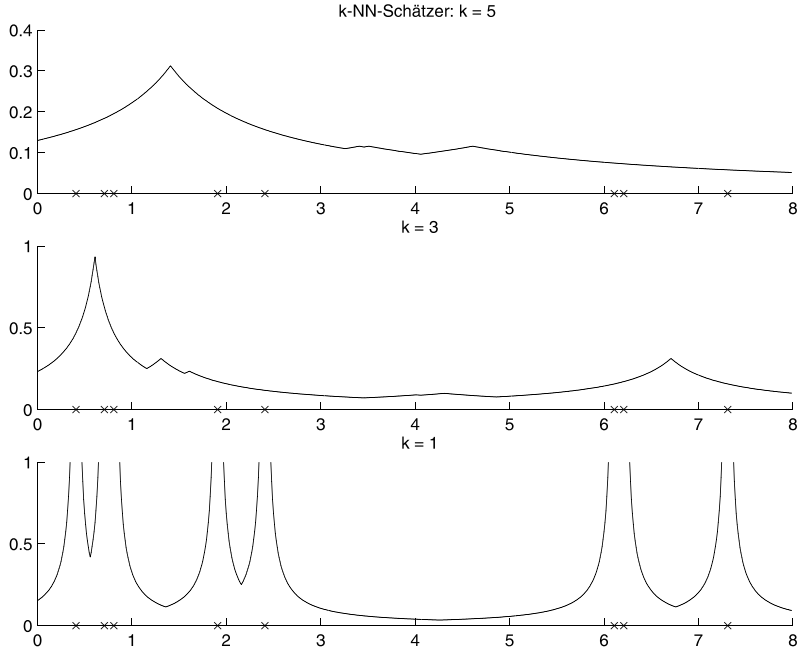


***Abb. 8.4*:** Schätzung durch die k-Nächste-Nachbarn-Methode für diverse k-Werte.

8.3 Verallgemeinerung auf multivariate Daten

Für eine gegebene Stichprobe von d-dimensionalen Beobachtungen $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ ist der multivariate Kernel-Dichteschätzer gleich

$$\hat{p}(\boldsymbol{x}) = \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) \tag{8.10}$$

mit der Bedingung, dass

$$\int_{\mathbb{R}^d} K(\boldsymbol{x}) d\boldsymbol{x} = 1\,.$$

Offensichtlich kommt der multivariate Gauß-Kernel in Frage:

$$K(\boldsymbol{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|\boldsymbol{u}\|^2}{2}\right]\,. \tag{8.11}$$

Fluch der Dimensionalität

Allerdings sollte man aufgrund des *Fluchs der Dimensionalität* bei der Verwendung von nichtparametrischen Schätzungen in hochdimensionalen Räumen Vorsicht walten lassen. Nehmen wir an, dass $\boldsymbol{x}$ achtdimensional ist und wir ein Histogramm mit zehn Intervallen pro Dimension nutzen; dann gibt es 10^8 Intervalle, von denen viele leer sein werden – es sei denn, wir haben eine große Anzahl an Daten – und deren Schätzungen gleich 0 sind. Bei hohen Dimensionen wird die Vorstellung von „nahe liegend" ebenfalls schwammiger, und somit sollte man vorsichtig bei der Wahl von h sein.

Beispielsweise impliziert die Verwendung der Euklidischen Werte in Gleichung 8.11, dass der Kernel in allen Dimensionen gleich skaliert ist. Wenn die Eingaben von unterschiedlicher Skalierung sind, sollten sie auf dieselbe Varianz hin normalisiert werden. Jedoch werden dadurch Korrelationen immer noch nicht in die Betrachtung einbezogen und bessere Ergebnisse lassen sich erzielen, wenn der Kernel dieselbe Form annimmt wie die zugrunde liegende Verteilung

$$K(\boldsymbol{u}) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}|^{1/2}} \exp\left[-\frac{1}{2}\boldsymbol{u}^T\mathbf{S}^{-1}\boldsymbol{u}\right] , \tag{8.12}$$

wobei $\mathbf{S}$ die Stichprobenkovarianzmatrix darstellt. Dies entspricht der Nutzung des Mahalanobis-Abstands statt der Euklidischen Distanz.

8.4 Nichtparametrische Klassifikation

Für die Klassifikation wird der nichtparametrische Ansatz benutzt, um die klassenbezogenen Dichten $p(\boldsymbol{x}|\mathcal{C}_i)$ zu schätzen. Der Kernel-Schätzer der klassenbezogenen Dichte ist gegeben als

$$\hat{p}(\boldsymbol{x}|\mathcal{C}_i) = \frac{1}{N_i h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t , \tag{8.13}$$

wobei r_i^t gleich 1 ist, falls $\boldsymbol{x}^t \in \mathcal{C}_i$, andernfalls ist es gleich 0. N_i ist die Anzahl an zugeordneten Instanzen, die zu $\mathcal{C}_i$ gehören: $N_i = \sum_t r_i^t$. Der Maximum-Likelihood-Schätzer der a-priori-Dichte ist $\hat{P}(\mathcal{C}_i) = N_i/N$. Somit kann die Diskriminante geschrieben werden als

$$\begin{aligned} g_i(\boldsymbol{x}) &= \hat{p}(\boldsymbol{x}|\mathcal{C}_i)\hat{P}(\mathcal{C}_i) \\ &= \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t \end{aligned} \tag{8.14}$$

und $\boldsymbol{x}$ wird der Klasse zugewiesen, für welche die Diskriminante ihr Maximum annimmt. Der gemeinsame Faktor $1/(Nh^d)$ kann ignoriert werden. Somit stimmt jede Trainingsinstanz für ihre Klasse und hat

keinen Effekt auf andere Klassen; das Gewicht dieses Votums wird durch die Kernel-Funktion $K(\cdot)$ gegeben, wobei typischerweise die näheren Instanzen stärker ins Gewicht fallen.

Für den speziellen Fall des k-NN-Schätzers haben wir

$$\hat{p}(\boldsymbol{x}|\mathcal{C}_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} , \tag{8.15}$$

wobei k_i die Anzahl an Nachbarn aus der Gruppe der k nächstliegenden ist, die zu $\mathcal{C}_i$ gehören; $V^k(\boldsymbol{x})$ ist das Volumen der d-dimensionalen um $\boldsymbol{x}$ zentrierten Hypersphäre mit Radius $r = \|\boldsymbol{x} - \boldsymbol{x}_{(k)}\|$, wobei $\boldsymbol{x}_{(k)}$ für die Beobachtung steht, die am k-nächsten zu $\boldsymbol{x}$ liegt (von allen Nachbarn aus allen Klassen von $\boldsymbol{x}$): $V^k = r^d c_d$ mit c_d als Volumen der Einheitssphäre in d Dimensionen, zum Beispiel $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3$, und so weiter. Somit ist

$$\hat{P}(\mathcal{C}_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|\mathcal{C}_i)\hat{P}(\mathcal{C}_i)}{\hat{p}(\boldsymbol{x})} = \frac{k_i}{k} . \tag{8.16}$$

k-NN-Klassifikator

Der *k-NN-Klassifikator* weist die Eingabe derjenigen Klasse mit den meisten Beispielen unter den k Nachbarn der Eingabe zu. Alle Nachbarn besitzen das gleiche Gewicht bei der Entscheidungsfindung und die Klasse mit der höchsten Anzahl an Treffern unter allen k Nachbarn wird gewählt. Gleichstände werden entweder willkürlich entschieden oder ein gewichtetes Votum wird genutzt. k wird normalerweise auf eine ungerade Zahl gesetzt, um die Zahl an Gleichständen zu minimieren: Uneindeutigkeiten bestehen generell in den Grenzregionen zwischen zwei benachbarten Klassen. Ein Spezialfall von k-NN ist der *Nächster-Nachbar-Klassifikator* mit $k = 1$, bei dem die Eingabe zur Klasse des nächstliegenden Musters zugewiesen wird. Dies zerlegt den Raum in der Form eines *Voronoi-Diagramms* (siehe Abbildung 8.5).

Nächster-Nachbar-Klassifikator

Voronoi-Diagramm

8.5 Verdichtete Nächste-Nachbarn-Methode

Speicher- und Rechenzeitkomplexität von nichtparametrischen Methoden sind proportional zur Größe des Trainingsdatensatzes, und *Verdichtungsmethoden* wurden vorgeschlagen, um die Anzahl an gespeicherten Instanzen zu verkleinern, ohne dabei die Leistungsfähigkeit zu reduzieren. Die Idee ist die, die kleinste Teilmenge $\mathcal{Z}$ von $\mathcal{X}$ auszuwählen, und zwar so, dass sich der Fehler nicht erhöht, wenn $\mathcal{Z}$ statt $\mathcal{X}$ genutzt wird (Dasarathy 1991).

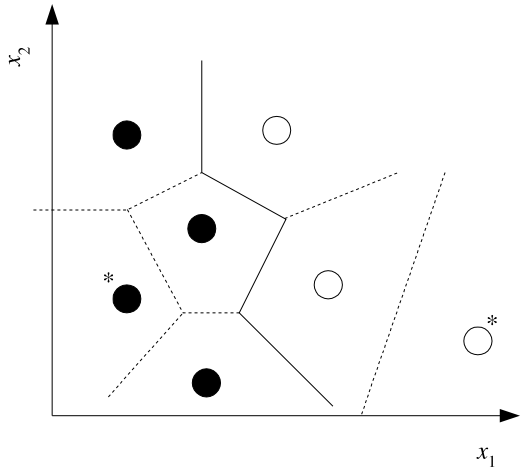


Abb. 8.5: Die gepunkteten Linien stellen das Voronoi-Diagramm dar, und die durchgezogene Linie ist die Klassendiskriminante. Beim verdichteten Nächste-Nachbarn-Algorithmus können diejenigen Instanzen, die nicht an der Definition der Diskriminante teilhaben (markiert durch „*“), entfernt werden, ohne dass der Fehler sich erhöht.

VERDICHTETER NÄCHSTE-NACHBARN-ALGORITHMUS

Die bekannteste und früheste Methode ist die des *verdichteten Nächste-Nachbarn-Algorithmus* (*condensed nearest neighbor*), bei dem 1-nn als nichtparametrischer Schätzer für die Klassifikation genutzt wird (Hart 1968). 1-nn approximiert die Diskriminante stückchenweise auf lineare Weise, und nur die Instanzen, welche die Diskriminante definieren, müssen behalten werden; eine Instanz innerhalb der Klassenregionen muss nicht gespeichert werden, da *ihr* nächster Nachbar derselben Klasse zugehört und sein Fehlen keinerlei Fehler (beim Trainingsdatensatz) verursacht (Abbildung 8.5). Solch eine Teilmenge wird als konsistente Teilmenge bezeichnet, und wir sind darum bemüht, die minimale konsistente Teilmenge zu finden.

Hart entwickelte einen Greedy-Algorithmus, um $\mathcal{Z}$ zu finden (Listing 8.1). Dieser Algorithmus beginnt mit einem leeren $\mathcal{Z}$ und überprüft dann, indem er die Instanzen in $\mathcal{X}$ zufällig eine nach der anderen passiert, ob sie durch 1-NN unter Nutzung der bereits in $\mathcal{Z}$ abgelegten Instanzen korrekt klassifiziert werden können. Wird eine Instanz falsch klassifiziert, wird sie $\mathcal{Z}$ hinzugefügt; findet eine korrekte Klassifikation statt, so bleibt $\mathcal{Z}$ unverändert. Der Trainingsdatensatz sollte mehrmals durchlaufen werden, bis keine weiteren Instanzen mehr hinzugefügt werden. Der Algorithmus vollführt eine lokale Suche, und in Abhängigkeit von der Reihenfolge, in der die Trainingsinstanzen betrachtet werden, können unterschiedliche Teilmengen gefunden werden, welche in ihrer Genauigkeit bei den Validierungsdaten möglicherweise differieren. Somit kann nicht garantiert werden, dass die minimale konsistente Teilmenge gefunden wird. Das Finden einer solchen optimalen Teilmenge ist NP-hart (Wilfong 1992).

Bei der verdichteten Nächste-Nachbarn-Methode handelt es sich um einen Greedy-Algorithmus, der darauf abzielt, den Trainingsfehler und die Komplexität zu minimieren, was durch die Größe der gespeicherten

```
Z ← ∅
Repeat
    For alle x ∈ X (in zufälliger Reihenfolge)
        Finde x' ∈ Z, so dass ‖x − x'‖ = min_{x^j ∈ Z} ‖x − x^j‖
        If Klasse(x) ≠ Klasse(x'), addiere x zu Z
Until Z ändert sich nicht
```

Listing 8.1: Verdichteter Nächste-Nachbarn-Algorithmus.

Teilmenge gemessen wird. Wir können eine erweiterte Fehlerfunktion schreiben:

$$E'(\mathcal{Z}|\mathcal{X}) = E(\mathcal{X}|\mathcal{Z}) + \lambda|\mathcal{Z}| \,, \tag{8.17}$$

wobei $E(\mathcal{X}|\mathcal{Z})$ der Fehler an $\mathcal{X}$ beim Speichern von $\mathcal{Z}$ ist. $|\mathcal{Z}|$ ist die Kardinalität von $\mathcal{Z}$ und der zweite Term straft die Komplexität. Wie bei jedem Regularisierungsschema repäsentiert λ den Kompromiss zwischen dem Fehler und der Komplexität, und zwar so, dass der Fehler für kleine λ wichtiger wird und bei wachsendem λ komplexe Modelle mehr gestraft werden. Die verdichtete Nächste-Nachbarn-Methode ist eine Methode zur Minimierung von Gleichung 8.17, jedoch sind auch andere Algorithmen zur Optimierung möglich.

8.6 Abstandsbasierte Klassifikation

Der k-Nächste-Nachbarn-Klassifikator ordnet jede Instanz derjenigen Klasse zu, der die meisten seiner Nachbarn angehören. Je ähnlicher sich zwei Instanzen sind, umso wahrscheinlicher ist es also, dass sie zu der gleichen Klasse gehören. Diesen Ansatz können wir auch bei der Klassifikation verwenden, vorausgesetzt wir haben ein geeignetes Ähnlichkeits- oder Abstandsmaß (Chen et al. 2009).

Die meisten Klassifikationsalgorithmen können zu einem abstandsbasierten Klassifikator umgeformt werden. In Abschnitt 5.5 haben wir zum Beispiel den parametrischen Ansatz mit Gauß-Klassen vorgestellt und in diesem Zusammenhang den Nächster-Mittelwert-Klassifikator eingeführt, wobei wir C_i gewählt haben, falls

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \min_{j=1}^{K} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_j) \,. \tag{8.18}$$

Im Falle von hypersphärischen Gauß-Funktionen, wo die Dimensionen unabhängig sind und die gleiche Größenordnung haben, ist das Abstandsmaß die Euklidische Distanz

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \|\boldsymbol{x} - \boldsymbol{m}_i\| \,.$$

Andernfalls ist es der Mahalanobis-Abstand

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = (\boldsymbol{x} - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x} - \boldsymbol{m}_i) \,,$$

wobei $\mathbf{S}_i$ die Kovarianzmatrix von C_i ist.

Beim semiparametrischen Ansatz, bei dem jede Klasse als eine Mischung aus Gauß-Funktionen geschrieben wird, können wir grob gesprochen sagen, dass wir C_i wählen, wenn von allen Clustermittelpunkten aller Klassen einer, der zu C_i gehört, am nächsten ist:

$$\min_{l=1}^{k_i} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{il}) = \min_{j=1}^{K} \min_{l=1}^{k_j} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{jl}) \,. \tag{8.19}$$

Dabei ist k_j die Anzahl der Cluster von C_j und $\boldsymbol{m}_{jl}$ bezeichnet den Mittelpunkt des Clusters l von C_j. Wiederum ist der Abstand entweder der Euklidische oder der Mahalanobis-Abstand, was von der Form der Cluster abhängt.

Der nichtparametrische Fall kann sogar noch flexibler sein: Anstatt ein Abstandsmaß pro Klasse oder Cluster zu haben, können wir für jede Nachbarschaft ein anderes haben, also für jedes kleine Gebiet des Eingaberaums. Mit anderen Worten, wir können *lokal adaptive Abstandsfunktionen* definieren, die dann bei der Klassifikation, beispielsweise mit k-NN, zum Einsatz kommen (Hastie und Tibshirani 1996; Domeniconi, Peng und Gunopulos 2002; Ramanan und Baker 2011).

Abstandslernen

Die Idee des *Abstandslernens* besteht darin, $\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\theta)$ zu parametrisieren, θ aus einer zugeordneten Stichprobe überwacht zu lernen und es dann mit k-NN zu verwenden (Bellet, Habrard und Sebban 2013). Der am weitesten verbreitete Ansatz benutzt den Mahalanobis-Abstand

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) = (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{M} (\boldsymbol{x} - \boldsymbol{x}^t) \tag{8.20}$$

mit der positiv definiten Matrix $\mathbf{M}$ als Parameter. Ein Beispiel ist der *Breiter-Margin-Nächster-Nachbar-Algorithmus* (engl. large margin nearest neighbor, Abk. *LMNN*) (Weinberger und Saul 2009), bei dem $\mathbf{M}$ so geschätzt wird, dass für alle Instanzen der Trainingsmenge der Abstand zu einem Nachbarn mit der gleichen Zuordnung stets kleiner ist als zu einem Nachbarn mit einer anderen Zuordnung. Wir diskutieren diesen Algorithmus ausführlich in Abschnitt 13.13.

LMNN-Algorithmus

Wenn die Eingabedimensionalität hoch ist, besteht ein Ansatz zur Vermeidung von Überanpassung darin, Seltenheitsbedingungen an $\mathbf{M}$ hinzuzufügen. Ein anderer Ansatz nutzt eine Approximation von niedrigem

Rang, bei dem $\mathbf{M}$ in $\mathbf{L}^T\mathbf{L}$ ($\mathbf{L}$ ist eine $k \times d$-Matrix mit $k < d$) faktorisiert wird. In diesem Fall haben wir

$$\begin{aligned}\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) &= (\boldsymbol{x} - \boldsymbol{x}^t)^T\mathbf{M}(\boldsymbol{x} - \boldsymbol{x}^t) = (\boldsymbol{x} - \boldsymbol{x}^t)^T\mathbf{L}^T\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t) \\ &= \big(\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t)\big)^T\big(\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t)\big) \\ &= (\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t)^T(\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t) \\ &= (\boldsymbol{z} - \boldsymbol{z}^t)^T(\boldsymbol{z} - \boldsymbol{z}^t) = \|\boldsymbol{z} - \boldsymbol{z}^t\|^2 , \end{aligned} \tag{8.21}$$

wobei $\boldsymbol{z} = \mathbf{L}\boldsymbol{x}$ die k-dimensionale Projektion von $\boldsymbol{x}$ ist und wir $\mathbf{L}$ anstatt $\mathbf{M}$ lernen. Wir sehen, dass der Mahalanobis-Abstand im d-dimensionalen Originalraum dem (quadrierten) Euklidischen Abstand im neuen k-dimensionalen Raum entspricht. Dies ist Ausdruck der dreifachen Beziehung zwischen Abstandsschätzung, Dimensionalitätsreduktion und Merkmalsextraktion: Das ideale Abstandsmaß ist definiert als der Euklidische Abstand in einem neuen Raum, dessen (Mindest-)Dimensionen aus den Originaleingaben auf die bestmögliche Art extrahiert wurde. Dies wird in Abbildung 8.6 veranschaulicht.

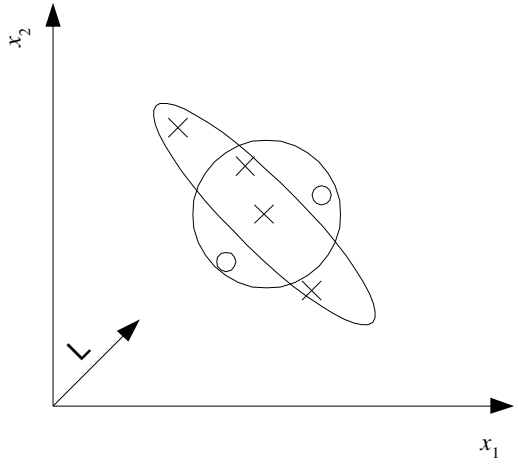


***Abb. 8.6*:** Mahalanobis-Abstand vs. Euklidischer Abstand bei der k-Nächster-Nachbar-Klassifikation. Es gibt zwei Klassen, die durch ∘ und × gekennzeichnet sind. Das fette × ist die Testinstanz und k ist 3. Punkte von gleichem Euklidischen Abstand definieren einen Kreis, der hier zur Fehlklassifikation führt. Wir sehen, dass es eine bestimmte Korrelationsstruktur gibt, die durch den Mahalanobis-Abstand erfasst werden kann; dieser definiert eine Ellipse und führt zu korrekter Klassifikation. Außerdem sehen wir, dass wir, wenn die Daten in die durch L angezeigte Richtung projiziert werden, in diesem reduzierten eindimensionalen Raum korrekt klassifizieren können.

Wenn die Eingaben diskret sind, können wir die *Hamming-Distanz* nutzen, welche die Anzahl an nicht übereinstimmenden Attributen zählt: HAMMING-DISTANZ

$$HD(\boldsymbol{x}, \boldsymbol{x}^t) = \sum_{j=1}^{d} 1(x_j \neq x_j^t) \,, \tag{8.22}$$

wobei

$$1(a) = \begin{cases} 1 \text{ falls } a \text{ wahr ist} \,, \\ 0 \text{ andernfalls} \,. \end{cases}$$

Dieses Konzept kann auch zusammen mit anwendungsspezifischen Ähnlichkeits- oder Abstandsmaßen verwendet werden. Beispiele sind spezialisierte Ähnlichkeits-/Abstandsmaße für übereinstimmende Bildteile beim Sehen, für übereinstimmende Sequenzen in der Bioinformatik und Maße für die Ähnlichkeit von Dokumenten bei der Sprachverarbeitung. Alle diese Maße können verwendet werden, ohne die entsprechenden Entitäten expliziert durch Vektoren darstellen zu müssen und ohne ein allgemeines Abstandsmaß wie den Euklidischen Abstand zu bemühen. In Kapitel 13 werden wir auf Kernel-Funktionen eingehen, die eine ähnliche Rolle spielen.

Wenn wir eine Funktion $\mathbf{S}(\boldsymbol{x}, \boldsymbol{x}^t)$ zur Ähnlichkeitsbewertung zweier Instanzen haben, können wir eine *ähnlichkeitsbasierte Darstellung* $\boldsymbol{x}'$ der Instanz $\boldsymbol{x}$ definieren, die durch den N-dimensionalen Vektor der Bewertungen mit allen Trainingsinstanzen $\boldsymbol{x}^t, t = 1, \ldots, N$ gegeben ist:

$$\boldsymbol{x}' = \left[s(\boldsymbol{x}, \boldsymbol{x}^1), s(\boldsymbol{x}, \boldsymbol{x}^2), \ldots, s(\boldsymbol{x}, \boldsymbol{x}^N) \right]^T \,.$$

Dieser Vektor kann anschließend von einem beliebigen Lerner benutzt werden (Pekalska und Duin 2002). Im Zusammenhang mit Kernel-Maschinen werden wir dies als *empirische Kernel-Abbildung* bezeichnen (Abschnitt 13.7).

8.7 Ausreißererkennung

Ausreißer, Neuheiten und *Anomalien* sind Instanzen, die sich sehr stark von allen übrigen Instanzen der Stichprobe unterscheiden. Ein Ausreißer kann ein Hinweis auf abnormes Verhalten im System sein. Beispielsweise kann ein Ausreißer in einem Datensatz von Kreditkartentransaktionen auf einen Betrug hindeuten. Bei Bilddaten im Bereich der medizinischen Diagnostik können Ausreißer Anomalien, etwa Tumore, anzeigen. Im Datenverkehr eines Netzwerks können sie die Spuren eines Hackerangriffs sein. Im Gesundheitswesen kann ein Ausreißer eine signifikante Abweichung vom normalen Patientenverhalten signalisieren. Ausreißer können aber auch durch Fehler bei der Datenaufzeichnung entstehen,

etwa durch fehlerhafte Sensoren. In diesem Fall sollten sie als Fehler erkannt und verworfen werden, um eine brauchbare Statistik zu bekommen.

AUSREISSER-ERKENNUNG

Die *Ausreißererkennung* lässt sich nicht gut als allgemeines, überwachtes Zweiklassen-Klassifikationsproblem interpretieren, bei dem typische Instanzen von Ausreißern zu separieren sind. Der Grund ist, dass es im Allgemeinen nur sehr wenige Instanzen gibt, die als Ausreißer einzuordnen sind, und diese wenigen Instanzen ergeben kein konsistentes Muster, das durch einen Zweiklassen-Klassifikator erfasst werden könnte. Stattdessen sind es bei der Ausreißererkennung die typischen Instanzen, die modelliert werden, weshalb man hier auch manchmal von einer *Einklassen-Klassifikation* spricht. Nachdem wir die typischen Instanzen modelliert haben, wird jede Instanz, die nicht in dieses Modell passt (wobei die Abweichung viele verschiedene Formen haben kann), als Anomalie angesehen. Ein weiteres allgemeines Problem besteht darin, dass die für das Training des Ausreißerdetektors verwendeten Daten nicht zugeordnet sind und daher Ausreißer gemischt mit typischen Instanzen enthalten können.

EINKLASSEN-KLASSIFIKATION

Ausreißererkennung bedeutet im Wesentlichen, all das aufzudecken, was im Normalfall nicht vorkommt, d. h., sie ist im Grunde eine Dichteschätzung mit anschließender Prüfung auf Instanzen, die unter dieser Schätzung für Dichte eine zu geringe Wahrscheinlichkeit haben. Wie üblich kann das angepasste Modell parametrisch, semiparametrisch oder nichtparametrisch sein. Im parametrischen Fall (Abschnitt 5.4) können wir zum Beispiel eine Gauß-Anpassung für die gesamte Datenmenge vornehmen, und jede Instanz, die eine geringe Wahrscheinlicheit hat (oder, gleichbedeutend, einen großen Mahalanobis-Abstand zum Mittelwert) ist ein potenzieller Ausreißer. Im semiparametrischen Fall (Abschnitt 7.2) passen wir zum Beispiel eine Kombination aus Gauß-Verteilungen an und prüfen, ob eine Instanz eine geringe Wahrscheinlicheit hat. Dies wäre eine Instanz, die entweder sehr weit von ihrem nächsten Clustermittelpunkt entfernt ist, oder eine, die für sich allein ein Cluster bildet.

Wenn die für die Anpassung des Modells verwendeten Daten selbst Ausreißer enthalten, ist es sinnvoller, einen nichtparametrischen Dichteschätzer zu verwenden, denn je parametrischer ein Modell ist, umso weniger robust ist es gegenüber Ausreißern. So kann es zum Beispiel passieren, dass ein einziger Ausreißer die Schätzung für den Mittelwert und die Kovarianz einer Gauß-Anpassung ernsthaft verschlechtert.

Wie wir aus der Diskussion in den vorherigen Abschnitten wissen, ist bei der nichtparametrischen Dichteschätzung die geschätzte Wahrscheinlicheit dort hoch, wo es viele Trainingsdaten in der Nähe gibt, und sie nimmt ab, wenn die Nachbarschaft zunehmend dünner besetzt ist. Ein Beispiel ist der *lokale Ausreißerfaktor*, der die Dichtheit der Nachbarschaft einer Instanz mit der mittleren Dichtheit der Nachbarschaften ihrer Nachbarn vergleicht (Breunig et al. 2000). Definieren wir $d_k(\boldsymbol{x})$ als den Abstand zwischen

LOKALER AUSREISSERFAKTOR

der Instanz $\boldsymbol{x}$ und ihrem k-nächsten Nachbarn. Außerdem definieren wir $\mathcal{N}(\boldsymbol{x})$ als die Menge der Trainingsinstanzen, die in der Nachbarschaft von $\boldsymbol{x}$ liegen, beispielsweise ihre k-nächsten Nachbarn. Betrachten wir nun den Abstand $\mathcal{N}(\boldsymbol{s})$ für $\boldsymbol{s} \in \mathcal{N}(\boldsymbol{x})$ und vergleichen wir $d_k(\boldsymbol{x})$ mit dem Mittel der $\mathcal{N}(\boldsymbol{s})$ für alle diese $\boldsymbol{s}$:

$$\text{LOF}(\boldsymbol{x}) = \frac{d_k(\boldsymbol{x})}{\sum_{\boldsymbol{s}\in\mathcal{N}(\boldsymbol{x})} d_k(\boldsymbol{s})/|\mathcal{N}(\boldsymbol{x})|} \,. \tag{8.23}$$

Wenn $\text{LOF}(\boldsymbol{x})$ nahe bei 1 liegt, ist $\boldsymbol{x}$ kein Ausreißer; wenn der Wert größer wird, dann steigt die Wahrscheinlicheit, dass es sich um einen Ausreißer handelt (siehe Abbildung 8.7).

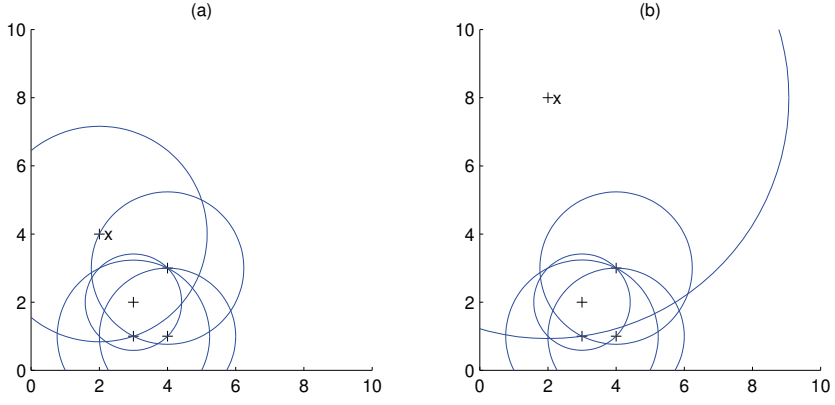


Abb. 8.7: Trainingsinstanzen sind durch ein + gekennzeichnet, × ist die Testinstanz und die Radien der um die Instanzen zentrierten Kreise entsprechen den Abständen zum jeweils drittnächsten Nachbarn. (a) LOF von × liegt nahe an 1 und die Testinstanz ist kein Ausreißer. (b) LOF von × ist viel größer als 1 und daher wahrscheinlich ein Ausreißer.

8.8 Nichtparametrische Regression: Glättungsmodelle

Bei der Regression und bei gegebenem Trainingsdatensatz $\mathcal{X} = \{x^t, r^t\}$ mit $r^t \in \mathbb{R}$ nehmen wir an, dass

$$r^t = g(x^t) + \epsilon \,.$$

Bei der parametrischen Regression gehen wir von einem Polynom eines bestimmten Grades aus und berechnen dessen Koeffizienten, welche die Summe des quadratischen Fehlers am Trainingsdatensatz minimieren. Die nichtparametrische Regression wird genutzt, wenn kein solches Polynom

vorausgesetzt werden kann; wir gehen lediglich davon aus, dass nahe liegende x auch nahe liegende Werte für $g(x)$ aufweisen. Wie bei der nichtparametrischen Dichteschätzung besteht bei gegebenem x unser Ansatz darin, die Nachbarschaft von x herauszufinden und den Durchschnitt der r Werte in der Nachbarschaft zu bilden, um $\hat{g}(x)$ zu berechnen. Der nichtparametrische Regressionsschätzer wird auch als *Glättungsfunktion* bezeichnet und bei der Schätzung spricht man von einer *Glättung* (Härdle 1990). Es existieren diverse Methoden zur Definition der Nachbarschaft und Durchschnittsbildung in selbiger, ähnlich wie bei der Dichteschätzung. Wir diskutieren die Methoden für univariate x. Wenn für die Dichteschätzung multivariate Kernel verwendet werden, ist es nicht kompliziert, die Methoden auf den multivariaten Fall zu erweitern.

GLÄTTUNGSFUNKTION

8.8.1 Gleitende Mittelwertglättung

Wenn wir einen Ursprung sowie eine Intervallbreite definieren und den Durchschnitt der r Werte im Intervall bilden, wie in den Histogrammen, dann erhalten wir ein *Regressogramm* (siehe Abbildung 8.8):

REGRESSOGRAMM

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} b(x, x^t) r^t}{\sum_{t=1}^{N} b(x, x^t)} , \tag{8.24}$$

wobei

$$b(x, x^t) = \begin{cases} 1 \text{ falls } x^t \text{ das gleiche Intervall mit } x \text{ hat} \\ 0 \text{ andernfalls} . \end{cases}$$

Das Vorhandensein von Diskontinuitäten an den Intervallgrenzen ist genauso störend wie der Zwang, einen Ursprung definieren zu müssen. Wie beim naiven Schätzer definieren wir bei der *gleitenden Mittelwertglättung* ein Intervall, das symmetrisch um x herum liegt, und bilden den Durchschnitt innerhalb von ihm (Abbildung 8.9).

GLEITENDE MITTELWERTGLÄTTUNG

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right)} \tag{8.25}$$

mit

$$w(u) = \begin{cases} 1 \text{ falls } |u| < 1 , \\ 0 \text{ andernfalls} . \end{cases}$$

Diese Methode ist besonders beliebt bei gleichmäßig im Raum verteilten Daten, beispielsweise bei Zeitreihen. Bei Anwendungen mit Rauschen kann man den Median von allen r^t im Intervall nutzen statt des Mittelwerts.

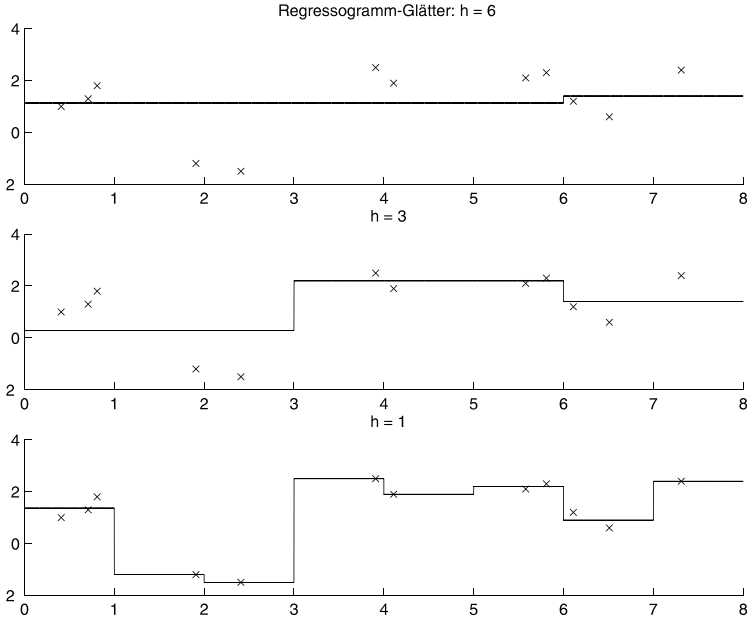


***Abb. 8.8*:** Regressogramme für diverse Intervallbreiten (× sind Datenpunkte).

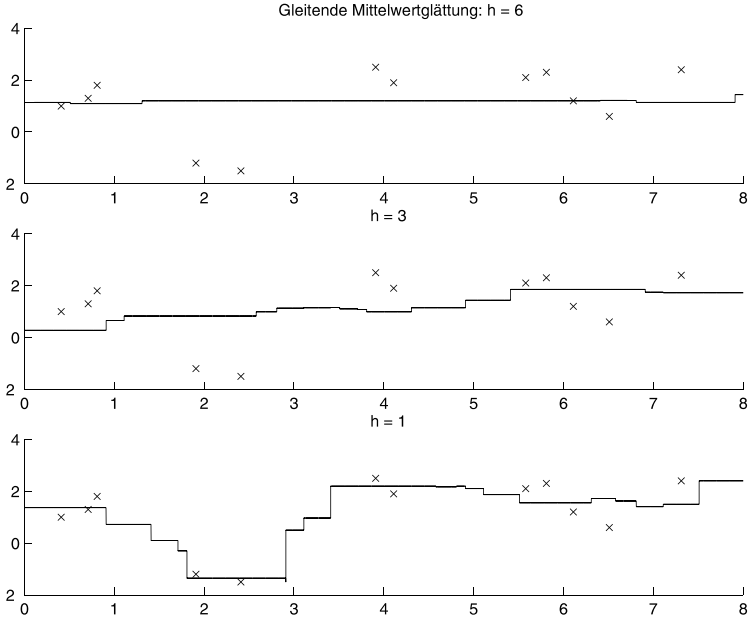


***Abb. 8.9*:** Gleitende Mittelwertglättung für diverse Intervallbreiten.

8.8.2 Glättung durch Kernel-Funktion

KERNEL-GLÄTTUNGS-FUNKTION

Wie beim Kernel-Schätzer können wir einen Kernel nutzen, der den entfernteren Punkten weniger Gewicht zuweist, und erhalten dadurch die *Kernel-Glättungsfunktion* (siehe Abbildung 8.10):

$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x - x^t}{h}\right) r^t}{\sum_t K\left(\frac{x - x^t}{h}\right)} . \tag{8.26}$$

k-NN-GLÄTTUNGS-FUNKTION

Typischerweise wird ein Gauß-Kernel $K(\cdot)$ genutzt. Statt h auf einen Wert zu fixieren, setzen wir k auf einen festen Wert, also die Anzahl an Nachbarn, und passen damit die Schätzung an die Dichte um x herum an und erhalten die *k-NN-Glättungsfunktion*.

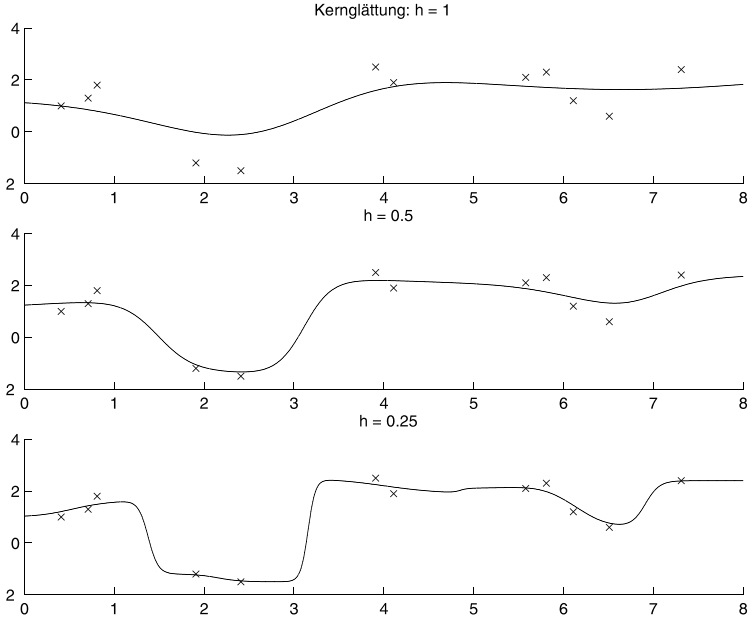


***Abb. 8.10*:** Kernel-Glättungsfunktion für diverse Intervallbreiten.

8.8.3 Gleitende Linienglättung

Statt einen Durchschnitt zu bilden und eine konstante Anpassung an einem Punkt zu berechnen, können wir auch noch einen weiteren Term in der Taylorschen Erweiterung einbeziehen und eine lineare Anpassung berechnen. Bei der sogenannten *gleitenden Linienglättung* können wir die Datenpunkte in der Nachbarschaft nutzen, so wie sie durch h oder

GLEITENDE LINIENGLÄTTUNG

k definiert sind, und eine lokale Regressionslinie anpassen (siehe Abbildung 8.11).

Bei der *lokal gewichteten gleitenden Linienglättung*, bekannt als *Loess*, nutzen wir statt einer harten Definition von Nachbarschaften lieber Kernel-Gewichtungen, so dass entfernte Punkte einen geringeren Effekt auf den Fehler haben.

LOKAL GEWICHTETE GLEITENDE LINIENGLÄTTUNG

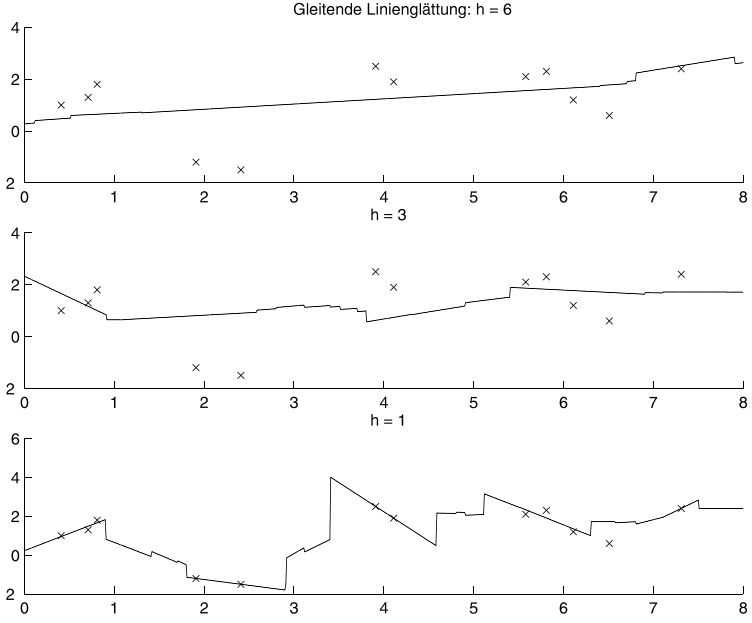


***Abb. 8.11*:** Gleitende Linienglättung für diverse Intervallbreiten.

8.9 Wahl des Glättungsparameters

Bei nichtparametrischen Methoden, ob für die Dichteschätzung oder die Regression, ist der kritische Parameter entweder der Glättungsparameter, wie er bei der Intervallbreite oder Kernel-Streubreite h benutzt wird, oder die Anzahl an Nachbarn k. Das Ziel besteht darin, eine Schätzung zu erhalten, die weniger variabel ist als die Datenpunkte. Wie wir bereits besprochen haben, ist eine Ursache für Variabilität in den Daten das Rauschen und die andere ist die Variabilität in der unbekannten zugrunde liegenden Funktion. Wir sollten gerade genug glätten, um den Effekt des Rauschens zu eliminieren – nicht mehr und nicht weniger. Mit einem zu großen h oder k tragen viele Instanzen zur Schätzung an einem Punkt bei und wir glätten ebenfalls die Variabilität aufgrund der Funktion, was in einer zu starken Glättung resultiert (Überglättung); bei zu kleinen k oder h haben einzelne Instanzen große Wirkung, wir glätten nicht einmal

das Rauschen und es entsteht eine zu geringe Glättung (Unterglättung). Mit anderen Worten, ein kleines h oder k führt zu kleiner Verzerrung aber großer Varianz. Ein größeres h oder k verringert die Varianz, aber erhöht die Verzerrung. Geman, Bienenstock und Doursat (1992) befassen sich mit Verzerrung und Varianz für nichtparametrische Schätzer.

Spline-Glättung

Diese Bedingung wird explizit in einer regularisierten Kostenfunktion codiert, wie sie bei der *Spline-Glättung* eingesetzt wird:

$$\sum_t \left[r^t - \hat{g}(x^t)\right]^2 + \lambda \int_a^b [\hat{g}''(x)]^2 dx \,. \tag{8.27}$$

Der erste Term ist der Fehler der Anpassung. $[a, b]$ ist der Eingabebereich; $\hat{g}''(\cdot)$ ist die *Krümmung* der geschätzten Funktion $\hat{g}(\cdot)$ und misst als solche die Variabilität. Somit straft der zweite Term sich schnell ändernde Schätzungen. λ bildet einen Kompromiss zwischen Variabilität und Fehler, wobei wir beispielsweise bei großem λ glattere Schätzungen erhalten.

Die Kreuzvalidierung wird genutzt, um optimale h, k oder λ zu bestimmen. Bei der Dichteschätzung wählen wir den Parameterwert, der die Likelihood des Validierungsdatensatzes maximiert. In einer überwachten Umgebung setzen wir eine Menge an in Frage kommenden Werten mit den Trainingsdaten ein (siehe Abbildung 8.12) und wählen den Parameterwert aus, der den Fehler am Validierungsdatensatz minimiert.

8.10 Anmerkungen

Die k-Nächste-Nachbarn-Methode und die kernelbasierte Schätzung wurden bereits vor sechzig Jahren entwickelt, doch aufgrund des hohen Bedarfs an Speicherplatz und Rechenaufwand gewann der Ansatz erst viel später an Popularität (Aha, Kibler und Albert 1991). Mit Fortschritten bei der Parallelverarbeitung und mit erschwinglicher werdendem Speicherplatz und Rechenaufwand hat sich die Nutzung solcher Methoden in letzter Zeit weiter verbreitet. Lehrbücher zur nichtparametrischen Schätzung sind Silverman (1986) und Scott (1992). Dasarathy (1991) bietet eine Sammlung vieler Aufsätze zu k-NN und dem Bearbeiten/Verkürzen von Regeln; ein weiterer Sammelband wurde 1997 von Aha herausgegeben.

Die nichtparametrischen Methoden können bei SIMD-Maschinen (Single Instruction Multiple Data) leicht parallelisiert werden; jeder Prozessor speichert eine Trainingsinstanz in seiner lokalen Speichereinheit und berechnet parallel den Wert der Kernel-Funktion für diese Instanz (Stanfill und Waltz 1986). Die Multiplikation mit einer Kernel-Funktion kann als Faltung betrachtet werden, und wir können die Fourier-Transformation nutzen, um die Schätzung effizienter zu berechnen (Silverman 1986). Es wurde auch bewiesen, dass die Spline-Glättung äquivalent zur Kernel-Glättung ist.

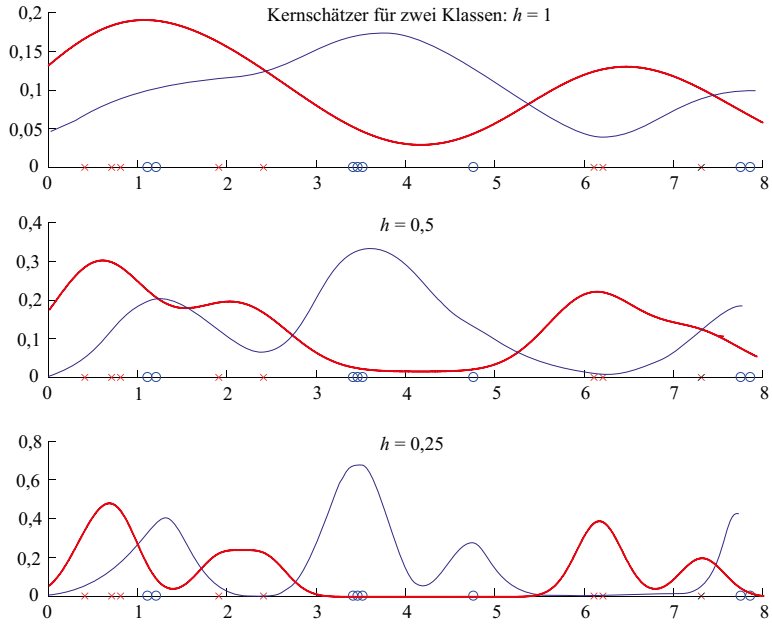


***Abb. 8.12*:** Kernel-Schätzung für verschiedene Intervalllängen für ein Zweiklassenproblem. Dargestellt sind die bedingten Dichten $p(x|C_i)$. Es scheint, dass die obere Schätzung zu stark glättet und die untere zu schwach, doch welche tatsächlich die beste ist, hängt davon ab, wo die Validierungsdatenpunkte liegen.

FALLBASIERTES SCHLIESSEN

Auf dem Gebiet der künstlichen Intelligenz wird der nichtparametrische Ansatz als *fallbasiertes Schließen* bezeichnet. Die Ausgabe wird gefunden, indem aus einigen bekannten vergangenen „Fällen“ interpoliert wird. Dies gestattet außerdem ein gewisses Maß an Wissensextraktion. Die gegebene Ausgabe kann gerechtfertigt werden, indem diese ähnlichen vergangenen Fälle aufgelistet werden.

Aufgrund seiner Einfachheit ist k-NN die am häufigsten genutzte nichtparametrische Klassifikationsmethode und erzielt in der Praxis recht ansehnlichen Erfolg in einer Vielzahl von Anwendungen. Sie hat die angenehme Eigenschaft, dass sie auch dann verwendet werden kann, wenn es nur sehr wenige zugeordnete Instanzen gibt. Beispielsweise kann es bei Anwendungen in der Forensik sein, dass wir nur ein Gesichtsbild pro Person haben.

Es wurde bewiesen (Cover und Hart 1967; neubetrachtet bei Duda, Hart und Stork 2001), dass in einem Fall mit großer Stichprobe und $N \to \infty$ das Risiko des nächsten Nachbarn ($k = 1$) nie schlechter ist, als das Zweifache des Bayesschen Risikos (welches das beste zu erzielende ist); in dieser Hinsicht sagt man, dass „die Hälfte der verfügbaren Informationen in

einer unendlichen Sammlung von klassifizierten Stichproben im nächsten Nachbarn enthalten sind“ (vgl. Cover und Hart 1967, 21). Im Falle von k-NN wurde bewiesen, dass sich das Risiko asymptotisch dem Bayesschen Risiko annähert, wenn k sich dem Unendlichen annähert.

Der kritischste Faktor bei der nichtparametrischen Schätzung ist die verwendete Distanzmetrik. Bei diskreten Attributen können wir schlicht die Hamming-Distanz nutzen, bei der wir lediglich die Anzahl an nichtübereinstimmenden Attributen summieren. Höher entwickelte Distanzfunktionen werden bei Wettschereck, Aha und Mohri (1997) sowie bei Webb (1999) diskutiert.

Das Gebiet der Abstandsschätzung bzw. des metrischen Lernens wird intensiv erforscht. In Bellet, Habrard und Sebban 2013 finden Sie dazu eine umfassende und aktuelle Übersicht. Die unterschiedlichen Einsatzmöglichkeiten von Ähnlichkeitsmaßen bei der Klassifikation werden in Chen et al. (2009) diskutiert. Beispiele für lokale Abstandsverfahren im Zusammenhang mit dem maschinellen Sehen werden in Ramanan und Baker 2011 vorgestellt.

Das Identifizieren von Ausreißern, Anomalien oder Neuheiten taucht als interessante Aufgabenstellung in verschiedenen Kontexten auf, etwa wenn es um das Entdecken von Fehlern oder Betrugsversuchen geht. Eine andere wichtige Anwendung ist das Erkennen signifikanter Abweichungen von früheren Daten, was zum Beispiel bei unzufriedenen Kunden vorkommen kann. Auch dieses Thema ist Gegenstand intensiver Forschung. Die Arbeiten von Hodge und Austin (2004) sowie von Chandola, Banerjee und Kumar (2009) bieten hierzu eine umfassende Übersicht.

Die nichtparametrische Regression wird im Detail bei Härdle (1990) behandelt. Hastie und Tibshirani (1990) diskutieren Glättungsmodelle und schlagen *additive Modelle* vor, bei denen eine multivariate Funktion als Summe univariater Schätzungen aufgefasst wird. Die lokal gewichtete Regression steht bei Atkeson, Moore und Schaal (1997) im Mittelpunkt. Diese Modelle sind den radialen Basisfunktionen und gemischten Expertensystemen sehr ähnlich, welche wir in Kapitel 12 behandeln werden.

ADDITIVE MODELLE

Bei der verdichteten Nächste-Nachbarn-Methode haben wir gesehen, dass es möglich ist, nur eine Teilmenge der Trainingsinstanzen zu behalten, nämlich diejenigen, die nahe an der Grenze liegen, und die Diskriminanz nur unter Verwendung dieser Teilmenge zu definieren. Diese Idee weist große Ähnlichkeit mit den *Support-Vektor-Maschinen* auf, die wir in Kapitel 13 behandeln. Dort diskutieren wir auch verschiedene Kernel-Funktionen, mit denen die Ähnlichkeit zwischen Instanzen gemessen werden kann, wobei wir uns unter anderem der Frage widmen, welches das beste dieser Maße ist. Die Vorhersage als Summe kombinierter Effekte der Trainingsinstanzen zu schreiben, ist auch die Idee, die den Gaußschen Prozessen (Kapitel 16) zugrunde liegt. In diesem Zusammenhang wird eine Kernel-Funktion als *Kovarianzfunktion* bezeichnet.

8.11 Übungen

1. Wie können wir zu einem glatten Histogramm gelangen?

   LÖSUNG: Wir können zwischen den beiden nächsten Intervallmittelpunkten interpolieren. Wir betrachten die Mittelpunkte als x^t und die Histogrammwerte als r^t und wenden ein beliebiges lineares oder kernelbasiertes Interpolationsschema an.

2. Beweisen Sie Gleichung 8.16.

   LÖSUNG: Gegeben ist

   $$\hat{p}(\boldsymbol{x}|\mathcal{C}_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \quad \text{und} \quad \hat{P}(\mathcal{C}_i) = \frac{N_i}{N} .$$

   Damit können wir schreiben

   $$\hat{P}(\mathcal{C}_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|\mathcal{C}_i)\hat{P}(\mathcal{C}_i)}{\sum_j \hat{p}(\boldsymbol{x}|\mathcal{C}_j)\hat{P}(\mathcal{C}_j)} = \frac{\frac{k_i}{N_i V^k(\boldsymbol{x})}\frac{N_i}{N}}{\sum_j \frac{k_j}{N_j V^k(\boldsymbol{x})}\frac{N_j}{N}} = \frac{k_i}{\sum_j k_j} = \frac{k_i}{k} .$$

3. Die parametrische Regression (Abschnitt 5.8) setzt Gaußsches Rauschen voraus und ist deshalb nicht robust gegenüber Ausreißern. Wie kann man das Verfahren robuster machen?

4. Wie können wir nach einer hierarchischen Clusteranalyse (Abschnitt 7.8) Ausreißer erkennen?

5. Wie verhält sich die verdichtete Nächste-Nachbarn-Methode für $k > 1$?

   LÖSUNG: Um im Falle $k > 1$ vollständige Genauigkeit ohne Fehlklassifikation zu erhalten, kann es notwendig sein, eine Instanz mehrfach zu speichern, so dass die korrekte Klasse die Mehrheit der Treffer erhält. Betrachten wir zum Beispiel $k = 3$ und nehmen wir an, dass $\boldsymbol{x}$ zwei Nachbarn hat, die zu einer anderen Klasse gehören. Dann müssen wir $\boldsymbol{x}$ zweimal speichern, damit, wenn $\boldsymbol{x}$ während des Tests gesehen wird, die Mehrheit (in diesem Fall zwei) von drei Nachbarn zu der richtigen Klasse gehört.

6. Bei der verdichteten Nächste-Nachbarn-Methode kann es sein, dass eine zuvor zu $\mathcal{Z}$ hinzugefügte Instanz nach einer späteren Hinzunahme nicht mehr gebraucht wird. Wie können wir solche nicht mehr benötigten Instanzen finden?

7. In einem Regressogramm kann man statt der Durchschnittswertbildung in einem Intervall und einer konstanten Anpassung auch die Instanzen nutzen, die in einem Intervall liegen und eine lineare Anpassung vornehmen (siehe Abbildung 8.13). Schreiben Sie den Code und vergleichen Sie diesen mit dem zugehörigen Regressogramm.

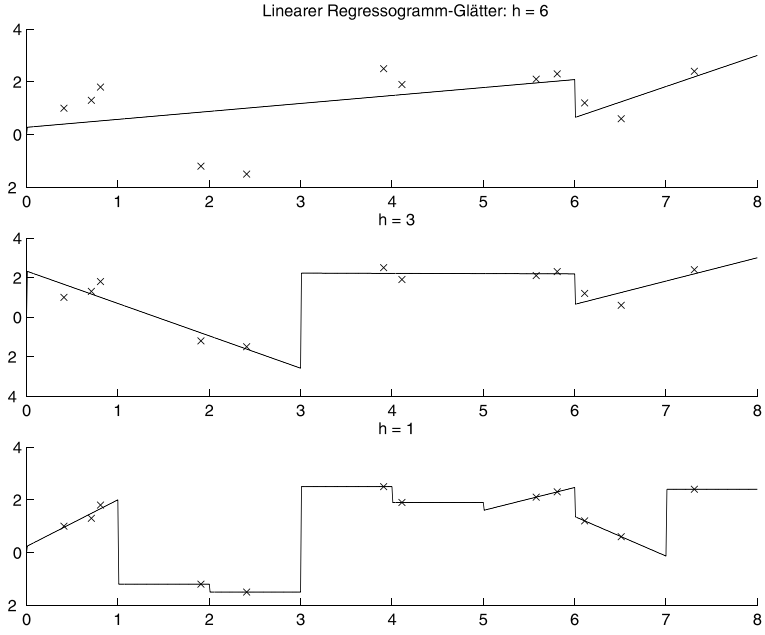


***Abb. 8.13*:** Regressogramme mit linearen Anpassungen der Intervalle für verschiedene Intervallbreiten.

8. Geben Sie die *Loess*-Fehlerfunktion an (Abschnitt 8.8.3).

   LÖSUNG: Die Ausgabe wird mithilfe eines linearen Modells $g(x) = ax + b$ berechnet, wobei wir im Zuge der gleitenden Linienglättung

   $$E(a, b|x, \mathcal{X}) = \sum_t w\left(\frac{x - x^t}{h}\right)[r^t - (ax^t + b)]^2 \quad \text{mit}$$

   $$w(u) = \begin{cases} 1 & \text{falls } |u| < 1 \\ 0 & \text{sonst} \end{cases}$$

   minimieren. Beachten Sie, dass wir hier nicht eine einzelne Fehlerfunktion haben, sondern für jede Testeingabe eine andere, die nur die am nächsten bei x liegenden Daten berücksichtigt und die an eine Linie in dieser Nachbarschaft angepasst wird.

   Loess ist die gewichtete Variante der gleitenden Linienglättung, wobei eine Kernel-Funktion $K(\cdot) \in (0, 1)$ die $w(\cdot) \in \{0, 1\}$ ersetzt:

   $$E(a, b|x, \mathcal{X}) = \sum_t K\left(\frac{x - x^t}{h}\right)[r^t - (ax^t + b)]^2 .$$

9. Entwickeln Sie eine inkrementelle Version der gleitenden Mittelwertglättung, welche wie die verdichtete Nächste-Nachbarn-Methode Instanzen nur dann speichert, wenn es wirklich nötig ist.

10. Verallgemeinern Sie die Kernel-Glättung auf multivariate Daten.

11. Bei der gleitenden Glättung können wir eine Konstante, eine Linie oder ein Polynom höheren Grades an einen Testpunkt anpassen. Wie können wir zwischen diesen unterscheiden?

    Lösung: Durch Kreuzvalidierung.

12. Können wir mit der gleitenden Mittelwertglättung, abgesehen davon, dass sie uns eine Schätzung liefert, auch ein Konfidenzintervall berechnen, das die Varianz (Unsicherheit) um die Schätzung in diesem Punkt anzeigt?

8.12 Literaturangaben

Aha, D. W., ed. 1997. Special Issue on Lazy Learning, *Artificial Intelligence Review* 11: issues 1–5.

Aha, D. W., D. Kibler, and M. K. Albert. 1991. „Instance-Based Learning Algorithm." *Machine Learning* 6: 37–66.

Atkeson, C. G., A. W. Moore, and S. Schaal. 1997. „Locally Weighted Learning." *Artificial Intelligence Review* 11: 11–73.

Bellet, A., A. Habrard, and M. Sebban. 2013. „A Survey on Metric Learning for Feature Vectors and Structured Data." *arXiv:1306.6709v2.*

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander. 2000. „LOF: Identifying Density-Based Local Outliers." In *ACM SIGMOD International Conference on Management of Data,* 93–104. New York: ACM Press.

Chandola, V., A. Banerjee, and V. Kumar. 2009. „Anomaly Detection: A Survey." *ACM Computing Surveys* 41 (3): 15:1–15:58.

Chen, Y., E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. 2009. „Similarity-Based Classification: Concepts and Algorithms." *Journal of Machine Learning Research* 11: 747–776.

Cover, T. M., and P. E. Hart. 1967. „Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory* 13: 21–27.

Dasarathy, B. V. 1991. *Nearest Neighbor Norms: NN Pattern Classification Techniques.* Los Alamitos, CA: IEEE Computer Society Press.

Domeniconi, C., J. Peng, and D. Gunopulos. 2002. „Locally Adaptive Metric Nearest-Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24: 1281–1285.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Geman, S., E. Bienenstock, and R. Doursat. 1992. „Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4: 1–58.

Härdle, W. 1990. *Applied Nonparametric Regression.* Cambridge, UK: Cambridge University Press.

Hart, P. E. 1968. „The Condensed Nearest Neighbor Rule." *IEEE Transactions on Information Theory* 14: 515–516.

Hastie, T. J., and R. J. Tibshirani. 1990. *Generalized Additive Models.* London: Chapman and Hall.

Hastie, T. J., and R. J. Tibshirani. 1996. „Discriminant Adaptive Nearest Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18: 607–616.

Hodge, V. J., and J. Austin. 2004. „A Survey of Outlier Detection Methodolo gies." Artificial Intelligence Review 22: 85–126.

Pekalska, E., and R. P. W. Duin. 2002. „Dissimilarity Representations Allow for Building Good Classifiers." *Pattern Recognition Letters* 23: 943–956.

Ramanan, D., and S. Baker. 2011. „Local Distance Functions: A Taxonomy, New Algorithms, and an Evaluation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33: 794–806.

Scott, D. W. 1992. *Multivariate Density Estimation.* New York: Wiley.

Silverman, B. W. 1986. *Density Estimation in Statistics and Data Analysis.* London: Chapman and Hall.

Stanfill, C. and D. Waltz. 1986. „Toward Memory-Based Reasoning." *Communications of the ACM* 29: 1213–1228.

Webb, A. 1999. *Statistical Pattern Recognition.* London: Arnold.

Weinberger, K. Q., and L. K. Saul. 2009. „Distance Metric Learning for Large Margin Classification." *Journal of Machine Learning Research* 10: 207–244.

Wettschereck, D., D. W. Aha, and T. Mohri. 1997. „A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms." *Artificial Intelligence Review* 11: 273–314.

Wilfong, G. 1992. „Nearest Neighbor Problems." *International Journal on Computational Geometry and Applications* 2: 383–416.

9 Entscheidungsbäume

Ein Entscheidungsbaum ist eine hierarchische Datenstruktur, durch welche die Teile-und-Herrsche-Strategie implementiert wird. Es handelt sich um eine effiziente nichtparametrische Methode, die sowohl für die Regression als auch für die Klassifikation genutzt werden kann. Wir diskutieren Lernalgorithmen, welche den Baum, ausgehend von einer gegebenen und mit Zuordnungen versehenen Trainingsstichprobe aufbauen, sowie die Frage, wie der Baum in eine Menge von einfachen und leicht verständlichen Regeln konvertiert werden kann. Eine weitere Möglichkeit ist das direkte Lernen von Regeln.

9.1 Einführung

Bei der parametrischen Schätzung definieren wir ein Modell über den gesamten Eingaberaum und lernen seine Parameter anhand aller Daten. Dann nutzen wir dasselbe Modell und dieselbe Parametermenge für beliebige Testeingaben. Bei der nichtparametrischen Schätzung teilen wir den Eingaberaum in lokale Regionen auf, die durch ein Distanzmaß wie die Euklidische Norm definiert werden, und für jede Eingabe wird das entsprechende lokale Modell genutzt, welches anhand der Daten in jener Region berechnet wurde. Bei den instanzbasierten Modellen, die wir in Kapitel 8 behandelt haben, und bei gegebener Eingabe ist die Identifikation der lokalen Daten, die das lokale Modell definieren, kostspielig; es erfordert die Berechnung der Entfernungen von der gegebenen Eingabe zu allen Instanzen, was $\mathcal{O}(N)$ entspricht.

Ein *Entscheidungsbaum* ist ein hierarchisches Modell für das überwachte Lernen, wobei die lokale Region in einer Sequenz von rekursiven Aufteilungen mit einer kleineren Anzahl an Schritten identifiziert wird. Ein Entscheidungsbaum besteht aus internen Entscheidungsknoten und terminalen Blättern (siehe Abbildung 9.1). Jeder *Entscheidungsknoten* m implementiert eine Testfunktion $f_m(\boldsymbol{x})$ mit diskreten Ergebnissen, die als Label für die Verzweigungen genutzt werden. Bei gegebener Eingabe wird an jedem Knoten ein Test durchgeführt und in Abhängigkeit des Ergebnisses eine der Verzweigungen ausgewählt. Dieser Prozess beginnt an der Wurzel und wird rekursiv solange wiederholt, bis ein *Blattknoten* erreicht wird; an diesem Punkt bildet der in dem Blatt stehende Wert die Ausgabe.

ENTSCHEIDUNGSBAUM

ENTSCHEIDUNGSKNOTEN

BLATTKNOTEN

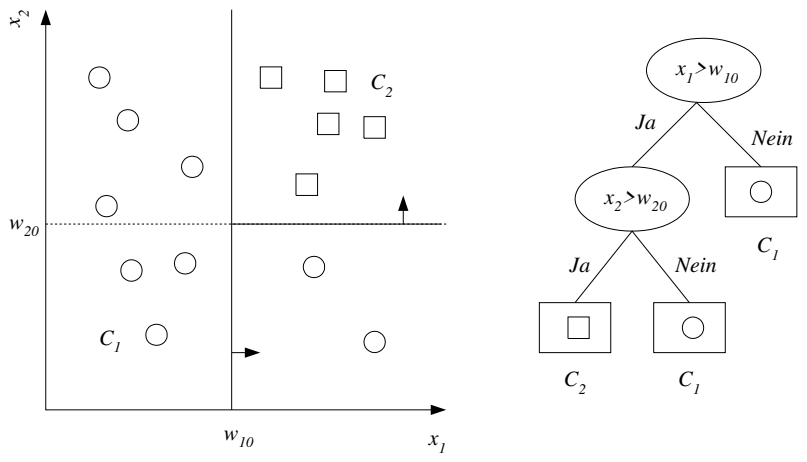


***Abb. 9.1*:** Beispiel eines Datensatzes und des entsprechenden Entscheidungsbaums. Ovale Knoten repräsentieren die Entscheidungsknoten, Vierecke stellen die Blattknoten dar. Der univariate Entscheidungsknoten teilt entlang einer Achse und nachfolgende Teilungen sind orthogonal zueinander. Nach der ersten Aufteilung besteht $\{\boldsymbol{x}|x_1 < w_{10}\}$ nur aus Instanzen einer Klasse und wird nicht weiter aufgeteilt.

Ein Entscheidungsbaum ist auch ein nichtparametrisches Modell in dem Sinne, dass wir keine parametrische Form für die Klassendichten annehmen und die Baumstruktur nicht von vornherein festgelegt ist. In Abhängigkeit von der den Daten inherenten Komplexität des Systems wird der Baum im Zuge des Lernens durch das Hinzufügen von Zweigen und Blättern wachsen.

Jedes $f_m(\boldsymbol{x})$ definiert eine Diskriminanzfunktion im d-dimensionalen Eingaberaum, wodurch dieser in kleinere Regionen zerlegt wird, welche wiederum weiter unterteilt werden, je weiter wir einem Pfad von der Wurzel nach unten folgen. $f_m(\cdot)$ ist eine einfache Funktion. So wird eine komplexe Aufteilungsfunktion, wenn sie als Entscheidungsbaum repräsentiert wird, in eine Sequenz einfacher Entscheidungen heruntergebrochen. Unterschiedliche Entscheidunsbaummethoden verwenden unterschiedliche Modelle für $f_m(\cdot)$, und die Modellklasse definiert sowohl die Form der Diskriminanzfunktion als auch die der Regionen. Jeder Blattknoten besitzt ein Ausgabelabel, welches im Falle der Klassifikation dem Klassencode und im Falle der Regression einem numerischen Wert entspricht. Ein Blattknoten definiert eine lokalisierte Region im Eingaberaum; Instanzen, die in solch eine Region fallen, haben die gleichen Zuordnungen (bei der Klassifikation) oder sehr ähnliche Ausgaben (bei der Regression). Die Grenzen der Regionen werden durch die Diskriminanzfunktionen bestimmt, die in den internen Knoten auf dem Pfad von der Wurzel zum Blattknoten codiert sind.

Die hierarchische Ordnung von Entscheidungen erlaubt die schnelle Lokalisierung der Region, die eine Eingabe abdeckt. Wenn die Entscheidungen beispielsweise binärer Natur sind, dann eliminiert jede Entscheidung im besten Fall die Hälfte der Fälle. Wenn es b Regionen gibt, dann kann im besten Fall die korrekte Region mit $\log_2 b$ Entscheidungen gefunden werden. Ein weiterer Vorteil des Entscheidungsbaums besteht in seiner Interpretierbarkeit. Wie wir in Kürze sehen werden, kann der Baum in eine Menge von leicht verständlichen *IF-THEN-Regeln* umgeformt werden. Aus diesem Grund sind Entscheidungsbäume sehr beliebt und werden gelegentlich exakteren aber schwieriger zu interpretierenden Methoden vorgezogen.

Wir beginnen mit univariaten Bäumen, bei denen ein Test an einem Entscheidungsknoten nur eine Eingabevariable nutzt, und wir werden betrachten, wie solche Bäume für die Klassifikation und Regression konstruiert werden können. Später verallgemeinern wir dies auf multivariate Bäume, bei denen alle Eingaben an einem internen Knoten verwendet werden können.

9.2 Univariate Bäume

Bei einem *univariaten Baum* umfasst der Test an jedem internen Knoten lediglich eine der Eingabedimensionen. Wenn es sich bei der genutzten Eingabedimension x_j um eine diskrete handelt, sie also einen von n möglichen Werten annimmt, überprüft der Entscheidungsknoten den Wert von x_j und wählt die entsprechende Verzweigung, wodurch eine n-fache Aufteilung implementiert wird. Wenn zum Beispiel ein Attribut für die Farbe steht und die möglichen Werte {rot, grün, blau} annehmen kann, dann hat ein Knoten zu diesem Attribut drei Verzweigungen, von denen jede einem der drei Werte für das Attribut entspricht.

UNIVARIATER BAUM

Ein Entscheidungsknoten hat diskrete Verzweigungen, und numerische Eingaben sollten diskretisiert werden. Wenn x_j numerisch (geordnet) ist, so besteht der Test aus einem Vergleich

$$f_m(\boldsymbol{x}) : x_j \geq w_{m0} \,, \tag{9.1}$$

wobei w_{m0} einen angemessen gewählten Schwellwert darstellt. Der Entscheidungsknoten zerlegt den Eingaberaum in zwei Teile: $L_m = \{\boldsymbol{x} | x_j \geq w_{m0}\}$ und $R_m = \{\boldsymbol{x} | x_j < w_{m0}\}$; hierbei spricht man von einer *binären Aufteilung*. Nachfolgende Entscheidungsknoten entlang eines Pfades von der Wurzel zu einem Blatt zerlegen diese wiederum jeweils in zwei Teile, indem sie andere Attribute nutzen. Die Blattknoten definieren Hyperrechtecke im Eingaberaum (siehe Abbildung 9.1).

BINÄRE AUFTEILUNG

Die Bauminduktion ist die Konstruktion eines Baums bei vorliegender Stichprobe. Für eine gegebene Menge existieren viele Bäume, welche diese ohne Fehler codieren; der Einfachheit halber interessiert uns der kleinste

unter ihnen, wobei die Baumgröße anhand der Menge der Knoten im Baum und der Komplexität der Entscheidungsknoten gemessen wird. Das Herausfinden des kleinsten Baumes ist NP-vollständig (Quinlan 1986), und wir sind gezwungen, auf Heuristiken basierende lokale Suchprozeduren zu nutzen, die brauchbare Bäume in angemessener Rechenzeit liefern.

Die Lernalgorithmen für Bäume sind vom Greedy-Typ. Beginnend an der Wurzel und mit den kompletten Daten suchen wir bei jedem Schritt die beste Aufteilung. Dadurch werden die Daten in zwei oder n Teile zerlegt, je nachdem, ob das gewählte Attribut numerisch oder diskret ist. Wir fahren dann mit der rekursiven Zerlegung für die entsprechende Teilmenge fort, bis keine Aufteilung mehr vonnöten ist; an dieser Stelle wird der Blattknoten erstellt und mit einem Label versehen.

9.2.1 Klassifikationsbäume

KLASSIFIKATIONSBAUM

Im Falle eines Entscheidungsbaums für die Klassifikation, also eines *Klassifikationsbaums*, wird die Güte einer Aufteilung durch ein *Verunreinigungsmaß* quantifiziert. Eine Aufteilung ist rein, wenn nach der Aufteilung für alle Verzweigungen gilt, dass alle Instanzen, die eine bestimmte Verzweigung wählen, derselben Klasse zugehörig sind. Sagen wir, dass für den Knoten m die Zahl an Instanzen, die Knoten m erreichen, durch N_m repräsentiert werden. Für den Wurzelknoten ist dies N. N_m^i von N_m gehören zur Klasse $\mathcal{C}_i$ mit $\sum_i N_m^i = N_m$. Wenn gegeben ist, dass eine Instanz den Knoten m erreicht, dann ist die Schätzung für die Wahrscheinlichkeit der Klasse $\mathcal{C}_i$

VERUNREINIGUNGSMASS

$$\hat{P}(\mathcal{C}_i|\boldsymbol{x}, m) \equiv p_m^i = \frac{N_m^i}{N_m} \,. \tag{9.2}$$

Knoten m ist rein, wenn p_m^i für alle i entweder 0 oder 1 sind. Der Wert 0 tritt auf, wenn keine der Instanzen, die Knoten m erreichen, der Klasse $\mathcal{C}_i$ zugehört, und der Wert 1 hingegen signalisiert, dass all diese Instanzen $\mathcal{C}_i$ angehören. Wenn die Aufteilung rein ist, müssen wir keine weitere Zerlegung vornehmen und können einen Blattknoten hinzufügen, der ein Label mit der Klasse erhält, für die p_m^i gleich 1 ist. Eine mögliche Funktion, um die Unreinheit zu messen, ist die *Entropie* (Quinlan 1986) (siehe Abbildung 9.2)

ENTROPIE

$$\mathcal{I}_m = -\sum_{i=1}^{K} p_m^i \log_2 p_m^i \,, \tag{9.3}$$

wobei $0 \log 0 \equiv 0$. Die Entropie spezifiziert in der Informationstheorie die minimale Anzahl an Bits, die nötig sind, um den Klassencode einer Instanz zu codieren. Wenn bei einem Zweiklassenproblem $p^1 = 1$ und $p^2 = 0$, so sind alle Beispiele $\mathcal{C}^1$ zugehörig, wir müssen nichts senden

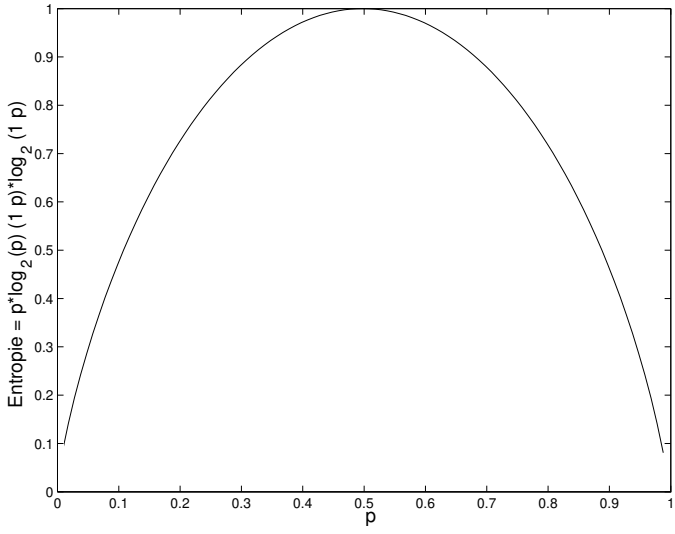


***Abb. 9.2*:** Die Entropiefunktion für ein Zweiklassenproblem.

und die Entropie ist gleich 0. Wenn $p^1 = p^2 = 0{,}5$, müssen wir ein Bit senden, um einen der zwei Fälle zu signalisieren und die Entropie ist gleich 1. Zwischen diesen zwei Extremen können wir uns Codierungen ausdenken und weniger als ein Bit pro Nachricht nutzen, indem wir kürzere Codierungen für die wahrscheinlicheren Klassen und längere Codierungen für die weniger wahrscheinlichen Klassen entwerfen. Wenn es $K > 2$ Klassen gibt, sind diese Aussagen ebenfalls zutreffend und die größte Entropie ist $\log_2 K$, wenn $p^i = 1/K$.

Die Entropie ist jedoch nicht das einzig mögliche Maß. Für ein Zweiklassenproblem, bei dem $p^1 \equiv p$ und $p^2 = 1 - p$, ist $\phi(p, 1-p)$ eine nichtnegative Funktion zur Messung der Unreinheit einer Aufteilung, wenn sie die folgenden Eigenschaften erfüllt (Devroye, Györfi und Lugosi 1996):

- $\phi(1/2, 1/2) \geq \phi(p, 1-p)$, für beliebige $p \in [0, 1]$.
- $\phi(0, 1) = \phi(1, 0) = 0$.
- $\phi(p, 1-p)$ erhöht sich in p in $[0, 1/2]$ und verringert sich in p in $[1/2, 1]$.

Beispiele sind:

1. Entropie

$$\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{9.4}$$

   Gleichung 9.3 ist die Generalisierung für $K > 2$ Klassen.

GINI-INDEX

2. *Gini-Index* (Breiman et al. 1984)

$$\phi(p, 1-p) = 2p(1-p) \tag{9.5}$$

3. Fehlklassifikationsfehler

$$\phi(p, 1-p) = 1 - \max(p, 1-p) \tag{9.6}$$

Diese können für $K > 2$ Klassen verallgemeinert werden, und der Fehlklassifikationsfehler kann bei Vorhandensein einer Verlustfunktion zum minimalen Risiko generalisiert werden (Übung 1). Die Forschung hat gezeigt, dass es keine signifikanten Unterschiede zwischen diesen drei Maßen gibt.

Falls die Instanzen, die einen Knoten m erreichen, nicht rein sind (alle der gleichen Klasse angehören), sollte diese Menge der Instanzen weiter aufgeteilt werden, um die Verunreinigung zu verringern. Dabei gibt es mehrere mögliche Attribute, anhand derer die Aufteilung durchgeführt werden kann. Für ein numerisches Attribut sind mehrere Auftrennungsgrenzen möglich. Wir suchen unter allen nach der Aufteilung, welche die Verunreinigung nach dieser Aufteilung minimiert, den wir wollen den kleinsten Baum generieren. Wenn die Teilmengen nach der Aufteilung reiner sind, werden im Nachhinein weniger zusätzliche Aufspaltungen (oder überhaupt keine) benötigt. Natürlich ist das nur lokal gesehen optimal und wir verfügen über keine Garantie, dass der kleinste Entscheidungsbaum gefunden wird.

Sagen wir, dass am Knoten m jetzt N_{mj} von N_m die Verzweigung j wählen; dies sind die $\boldsymbol{x}^t$, für welche der Test $f_m(\boldsymbol{x}^t)$ das Ergebnis j liefert. Für ein diskretes Attribut mit n Werten gibt es n Ergebnisse, und für ein numerisches Attribut können zwei mögliche Ergebnisse ($n = 2$) eintreten; in jedem Fall gilt $\sum_{j=1}^{n} N_{mj} = N_m$. Die N_{mj}^i aus N_{mj} gehören zur Klasse $\mathcal{C}_i$: $\sum_{i=1}^{K} N_{mj}^i = N_{mj}$. Ähnlich dazu gilt: $\sum_{j=1}^{n} N_{mj}^i = N_m^i$.

Wenn dann bekannt ist, dass der Test am Knoten m das Ergebnis j liefert, so ist die Schätzung für die Wahrscheinlichkeit der Klasse $\mathcal{C}_i$ gleich

$$\hat{P}(\mathcal{C}_i|\boldsymbol{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \tag{9.7}$$

und die gesamte Verunreinigung nach der Aufteilung ergibt sich aus

$$\mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \log_2 p_{mj}^i \ . \tag{9.8}$$

Um im Falle eines numerischen Attributes in der Lage zu sein, p_{mj}^i mit Hilfe von Gleichung 9.1 zu berechnen, müssen wir auch w_{m0} für jenen Knoten kennen. Es existieren $N_m - 1$ mögliche w_{m0} zwischen N_m

Datenpunkten. Wir müssen aber nicht alle (möglicherweise unendlich viele) Punkte testen; es genügt beispielsweise, auf halbem Wege zwischen Punkten zu testen. Man beachte auch, dass die beste Aufteilung immer zwischen nebeneinanderliegenden Punkten mit unterschiedlicher Klassenzugehörigkeit zu finden ist. Daher probieren wir diese, und die beste hinsichtlich der Reinheit wird dann auch für die Reinheit des Attributs gewählt. Im Falle eines diskreten Attributs ist solch eine Iteration nicht nötig.

Somit berechnen wir für alle Attribute (diskrete als auch numerische) bzw. bei einem numerischen Attribut für alle Aufteilungspositionen die Unreinheit und wählen den Wert mit beispielsweise der geringsten Entropie, wie in Gleichung 9.8 gezeigt wurde. Dann wird die Baumkonstruktion rekursiv und parallel für die unreinen Verzweigungen fortgesetzt, bis alle rein sind. Dies ist die Grundlage des CART-Algorithmus (engl. *classification and regression trees,*) (Breiman et al. 1984), des *ID3-Algorithmus* (Quinlan 1986) und seiner Erweiterung *C4.5* (Quinlan 1993). Der Pseudocode des Algorithmus ist im Listing 9.1 dargestellt.

CART-Algorithmus

ID3-Algorithmus C4.5

```
GeneriereBaum(𝒳)
    If KnotenEntropie(𝒳)< θ_I /* Gleichung 9.3 */
      Erstelle Blatt mit Label der häufigsten Klasse in 𝒳
      Return
    i ← AufspaltungsAttribut(𝒳)
    For jede Verzweigung von x_i
      Finde 𝒳_i, das in Verzweigung liegt
      GeneriereBaum(𝒳_i)

AufspaltungsAttribut(𝒳)
    MinEnt← MAX
    For alle Attribute i = 1,...,d
        If x_i ist diskret mit n Werten
          Teile 𝒳 in 𝒳_1,...,𝒳_n durch x_i auf
          e ← AufspaltungsEntropie(𝒳_1,...,𝒳_n) /* Gleichung 9.8 */
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i ist numerisch */
          For alle möglichen Aufspaltungen
              teile Aufspaltung 𝒳 in 𝒳_1, 𝒳_2 an x_i
              e← AufspaltungsEntropie(𝒳_1, 𝒳_2)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

***Listing 9.1*:** Konstruktion eines Klassifikationsbaums.

Es kann hinzugefügt werden, dass wir bei jedem Schritt während der Konstruktion des Baumes die Aufteilung wählen, welche die deutlichste Abnahme an Verunreinigung zur Folge hat. Das entspricht dem Unter-

schied zwischen der Verunreinigung von Daten, die Knoten m erreichen (Gleichung 9.3), und der gesamten Entropie von Daten, die seine Verzweigungen nach der Aufteilung erreichen (Gleichung 9.8).

Ein Problem besteht darin, dass diese Art der Zerlegung Attribute mit vielen Werten bevorzugt. Wenn viele Werte existieren, gibt es viele Verzweigungen und die Verunreinigung kann nach der Aufteilung deutlich geringer ausfallen. Wenn wir beispielsweise den Index t der Trainingsinstanzen als Attribut nehmen, dann wird das Maß der Unreinheit dieses auswählen, denn dann ist die Verunreinigung einer jeden Verzweigung gleich 0, obwohl es sich nicht um ein sinnvolles Merkmal handelt. Knoten mit vielen Verzweigungen sind komplex und widerstreben unserer Idee, die Klassendiskriminanten in einfache Entscheidungen zu zerlegen. Methoden wurden entwickelt, um solche Attribute zu strafen und die deutliche Abnahme der Verunreinigung sowie den Verzweigungsfaktor auszubalancieren.

Wenn Rauschen existiert, kann es vorkommen, dass wir beim Aufbauen des Baumes bis zum besten Reinheitsgrad einen sehr großen Baum erstellen und dieser zu stark angepasst ist. Man betrachte zum Beispiel den Fall einer Instanz mit falschem Label inmitten einer Gruppe mit korrekt ausgewiesenen Instanzen. Um eine derartige Überanpassung abzufangen, wird die Baumkonstruktion beendet, wenn die Knoten rein genug sind, sprich, eine Teilmenge der Daten wird nicht weiter zerlegt, wenn $\mathcal{I} < \theta_I$. Dies impliziert, dass wir nicht fordern, dass p_{mj}^i exakt 0 oder 1 ist, sondern nur nahe genug an einem der beiden Werte liegen muss, und zwar mit einem Schwellwert θ_p. In solch einem Fall wird ein Blattknoten erstellt und der Klasse zugeordnet, für welche p_{mj}^i am größten ist.

θ_I (oder θ_p) ist der Komplexitätsparameter, wie h oder k von der nichtparametrischen Schätzung. Wenn sie klein sind, so ist die Varianz groß und der Baum muss deutlich an Höhe gewinnen, um den Datensatz akkurat zu repräsentieren; sind sie jedoch groß, so ist die Varianz niedriger und ein kleinerer Baum repräsentiert im groben den Datensatz, könnte unter Umständen jedoch durch starke Verzerrung geprägt sein. Der Idealwert hängt von den Folgen von Fehlklassifikationen ab sowie von den Kosten für Speicherplatz und Verarbeitung.

Es ist generell ratsam, in einem Blatt die a-posteriori-Wahrscheinlichkeiten von Klassen zu speichern, statt ein Blatt nur der Klasse mit dem höchsten a-posteriori-Wert zuzuordnen. Diese Wahrscheinlichkeiten könnten in späteren Schritten vonnöten sein, beispielsweise bei der Risikokalkulation. Man beachte, dass wir nicht die Instanzen, die einen Knoten erreichen, oder die genaue Anzahl speichern müssen; die Verhältnisse sind ausreichend.

9.2.2 Regressionsbäume

Ein *Regressionsbaum* wird auf fast dieselbe Weise konstruiert wie ein Klassifikationsbaum, außer, dass das für die Klassifikation angemessene Verunreinigungsmaß durch ein für die Regression angemessenes Maß ersetzt wird. Nehmen wir an, für Knoten m sei $\mathcal{X}_m$ die Teilmenge von $\mathcal{X}$, die Knoten m erreichen, sprich es handelt sich um die Menge aller $\boldsymbol{x} \in \mathcal{X}$, welche alle Bedingungen an den Entscheidungsknoten auf dem Pfad von der Wurzel zu Knoten m erfüllen. Wir definieren

REGRESSIONSBAUM

$$b_m(\boldsymbol{x}) = \begin{cases} 1 \text{ falls } \boldsymbol{x} \in \mathcal{X}_m\text{: } \boldsymbol{x} \text{ erreicht Knoten } m\,, \\ 0 \text{ andernfalls}\,. \end{cases} \tag{9.9}$$

Bei der Regression wird die Güte einer Aufteilung durch den mittleren quadratischen Fehler von dem geschätzten Wert gemessen. Sei g_m der geschätzte Wert am Knoten m.

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\boldsymbol{x}^t)\,, \tag{9.10}$$

wobei $N_m = |\mathcal{X}_m| = \sum_t b_m(\boldsymbol{x}^t)$.

An einem Knoten nutzen wir den Mittelwert (den Median, wenn zuviel Rauschen vorhanden ist) der zu erzielenden Ausgaben von Instanzen, die den Knoten erreichen,

$$g_m = \frac{\sum_t b_m(\boldsymbol{x}^t) r^t}{\sum_t b_m(\boldsymbol{x}^t)}\,. \tag{9.11}$$

Somit entspricht Gleichung 9.10 der Varianz bei m. Wenn an einem Knoten der Fehler akzeptabel ist, d. h. wenn $E_m < \theta_r$ gilt, dann wird ein Blattknoten erstellt, welcher den Wert von g_m speichert. Ebenso wie beim Regressogram aus Kapitel 8 wird auch hier stückchenweise eine konstante Approximation mit Diskontinuitäten an den Blattgrenzen erschaffen.

Ist der Fehler inakzeptabel, dann werden die den Knoten m erreichenden Daten weiter zerlegt, so dass die Summe der Fehler in den Verzweigungen minimal ist. Wie bei der Klassifikation suchen wir an jedem Knoten nach dem Attribut (und Aufteilungsgrenzwert für ein numerisches Attribut), welches den Fehler minimiert, und fahren dann rekursiv fort.

Definieren wir $\mathcal{X}_{mj}$ als die Teilmenge von $\mathcal{X}_m$ und nehmen die Verzweigung j: $\cup_{j=1}^n \mathcal{X}_{mj} = \mathcal{X}_m$. Wir definieren:

$$b_{mj}(\boldsymbol{x}) = \begin{cases} 1 \text{ falls } \boldsymbol{x} \in \mathcal{X}_{mj}\text{: } \boldsymbol{x} \text{ erreicht Knoten } m \\ \qquad\qquad\qquad\qquad \text{und wählt Verzweiung } j, \\ 0 \text{ andernfalls}\,. \end{cases} \tag{9.12}$$

g_{mj} ist der geschätzte Wert in Verzweigung j von Knoten m.

$$g_{mj} = \frac{\sum_t b_{mj}(\boldsymbol{x}^t) r^t}{\sum_t b_{mj}(\boldsymbol{x}^t)} \tag{9.13}$$

Der Fehler nach der Aufteilung berechnet sich als:

$$E'_m = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\boldsymbol{x}^t) \; . \tag{9.14}$$

Die Verringerung des Fehlers für eine beliebige Aufteilung ergibt sich aus der Differenz zwischen Gleichung 9.10 und Gleichung 9.14. Wir suchen nach der Aufteilung, durch den diese Verringerung maximal wird, beziehungsweise äquivalent dazu, durch den Gleichung 9.14 ihr Minimum annimmt. Der im Listing 9.1 angegebene Code kann für das Üben eines Regressionsbaumes adaptiert werden, indem die Entropieberechnungen mit dem mittleren quadratischen Fehler und die Klassenzuordnungen mit den Durchschnittswerten ersetzt werden.

Der mittlere quadratische Fehler ist eine mögliche Fehlerfunktion; eine andere findet sich im schlimmstmöglichen Fehler,

$$E_m = \max_j \max_t |r^t - g_{mj}| b_m(\boldsymbol{x}^t) \; , \tag{9.15}$$

mit dem wir garantieren können, dass der Fehler für eine beliebige Instanz nie größer als ein festgelegter Schwellwert ist.

Der akzeptable Fehlerschwellwert ist der Komplexitätsparameter; ist er niedrig, so generieren wir große Bäume und riskieren eine Überanpassung; ist er groß, so verursachen wir eine Unteranpassung und glätten zu stark (siehe Abbildung 9.3 und 9.4).

Ähnlich dem Übergang vom gleitenden Mittelwert zur gleitenden Linie bei der nichtparametrischen Regression können wir statt der Durchschnittswertbildung an einem Blatt, also statt einer konstanten Anpassung, auch eine lineare Regressionsanpassung über die Instanzen vornehmen, die das Blatt wählen:

$$g_m(\boldsymbol{x}) = \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} \; . \tag{9.16}$$

Dies verursacht eine Abhängigkeit zwischen der Schätzung in einem Blatt und $\boldsymbol{x}$ und generiert kleinere Bäume, jedoch auf Kosten von zusätzlichem Berechnungsaufwand an einem Blattknoten.

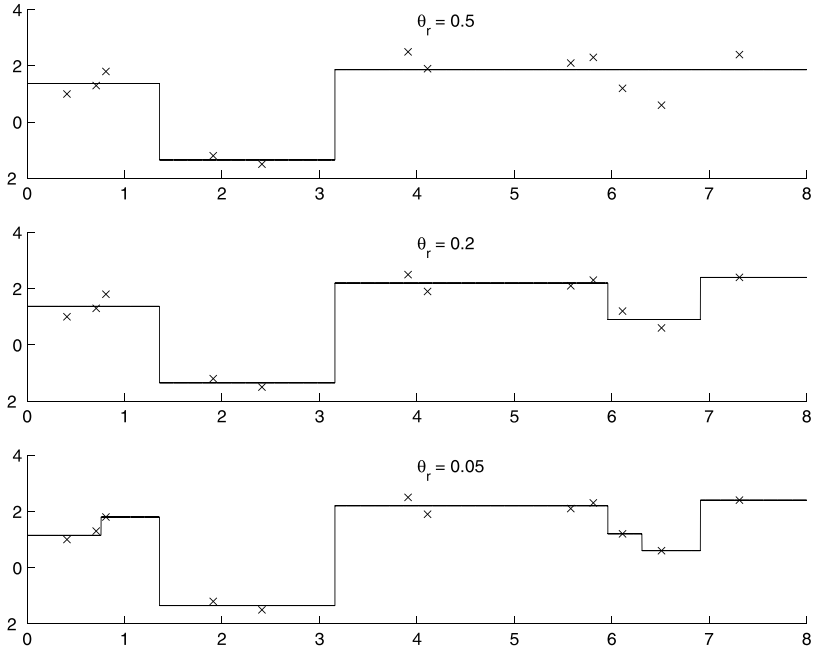


***Abb. 9.3*:** Glättungen von Regressionsbäumen für verschiedenste Werte von θ_r. Die entsprechenden Bäume finden sich in Abbildung 9.4.

9.3 Pruning

Häufig wird die Menge der Instanzen, die einen Knoten erreichen, nicht weiter zerlegt, wenn die Anzahl an Instanzen, die den Knoten erreichen, kleiner ist als ein gewisser Prozentsatz des Datensatzes, beispielsweise 5 Prozent, unabhängig von der Verunreinigung oder dem Fehler. Der Gedanke ist der, dass eine jede Entscheidung, die auf zu wenigen Instanzen basiert, Varianz und somit einen Generalisierungsfehler verursacht. Beim Abbruch der Baumkonstruktion noch bevor diese vollständig ist spricht man vom *Prepruning* des Baumes (*Pruning*: dt. *Beschneidung* oder *Stutzung*).

PREPRUNING

Eine weitere Möglichkeit, einfachere Bäume zu erhalten, besteht im *Postpruning*, was in der Praxis bessere Ergebnisse erzielt als das Prepruning. Wir haben bereits gesehen, dass die Konstruktion eines Baumes zur Greedy-Familie gehört und wir bei jedem Schritt eine Entscheidung treffen, also einen Entscheidungsknoten generieren. Dann fahren wir fort, ohne bereits getroffene Entscheidungen jemals wieder zu überprüfen und für diese eventuell eine Alternative auszuprobieren. Die einzige Ausnahme stellt das Postpruning dar, bei dem wir versuchen, unnötige Unterbäume zu finden und zu entfernen.

POSTPRUNING

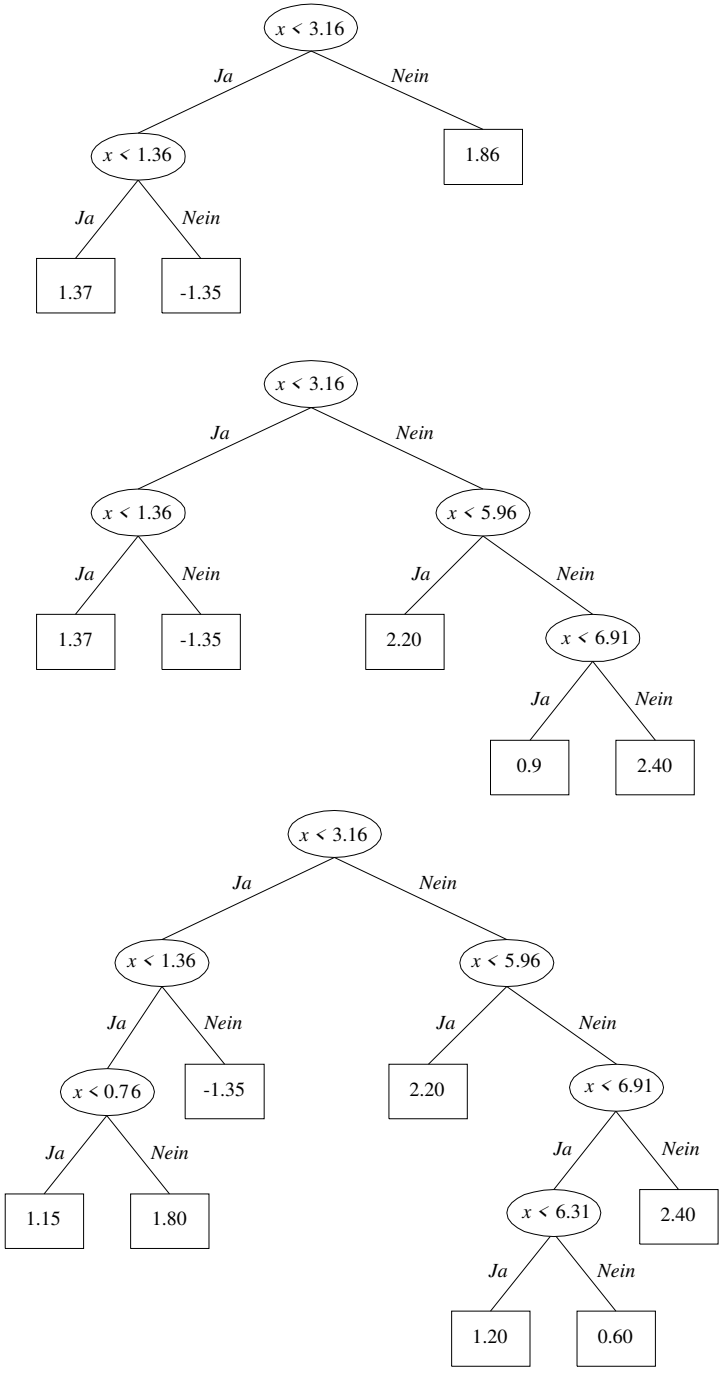


***Abb. 9.4*:** Regressionsbäume, welche die Glättungen aus Abbildung 9.3 für diverse Werte von θ_r implementieren.

Beim Postpruning bauen wir den Baum auf, bis all seine Blätter rein sind und der Fehler gleich null ist. Dann finden wir die Unterbäume heraus, die eine Überanpassung verursachen und entfernen diese. Von der ursprünglichen mit Labeln versehenen Stichprobe legen wir eine *Pruning-Menge* beiseite, die beim Üben ungenutzt bleibt. Jeden Unterbaum ersetzen wir durch einen Blattknoten, der ein Label (je nachdem ob für die Klassifikation oder Regression) entsprechend der Instanzen, die durch den Unterbaum abgedeckt sind, trägt. Liefert diese Ersetzung keine schlechteren Ergebnisse bezogen auf die Pruning-Menge, so behalten wir den Blattknoten anstatt des Unterbaums bei, da es keinen Grund für die zusätzliche Komplexität des Unterbaums gibt. Andernfalls behalten wir den Unterbaum bei.

PRUNING-MENGE

Beim dritten Baum in Abbildung 9.4 gibt es beispielsweise einen Unterbaum, der mit der Bedingung $x < 6{,}31$ beginnt. Dieser Unterbaum kann durch einen Blattknoten mit dem Label $y = 0{,}9$ ersetzt werden (wie im zweiten Baum), wenn der Fehler an der Pruning-Menge sich dadurch nicht erhöht. Es ist jedoch zu beachten, dass die Pruning-Menge nicht mit der Validierungsmenge (von der sie sich unterscheidet) verwechselt werden darf.

Beim Vergleich von Prepruning und Postpruning stellen wir fest, dass das Prepruning zwar schneller vonstatten geht, das Postpruning jedoch im Allgemeinen zu genaueren Bäumen führt.

9.4 Regelextraktion aus Bäumen

Ein Entscheidungsbaum nimmt seine eigene Merkmalsextraktion vor. Der univariate Baum nutzt lediglich die benötigten Variablen, und es kann sogar vorkommen, dass nach der Erstellung des Baums einige Merkmale überhaupt keine Verwendung finden. Wir können also sagen, dass Merkmale näher an der Wurzel von größerer globaler Wichtigkeit sind. Der in Abbildung 9.5 dargestellte Entscheidungsbaum beispielsweise nutzt x_1, x_2 und x_4, nicht aber x_3. Es besteht die Möglichkeit, einen Entscheidungsbaum für die Merkmalsextraktion zu nutzen. Dazu bauen wir den Baum auf und nutzen als Eingabe für eine sich anschließende Lernmethode nur diejenigen Merkmale, die auch vom Baum benutzt werden.

Ein weiterer wichtiger Vorteil von Entscheidungsbäumen besteht in ihrer *Interpretierbarkeit*. Die Entscheidungsknoten tragen leicht verständliche Bedingungen. Jeder Pfad von der Wurzel zu einem Blatt entspricht einer Verkettung von Tests, da all diese Bedingungen erfüllt sein sollten, um das Blatt zu erreichen. Zusammen können diese Pfade als eine Menge von *IF-THEN-Regeln* niedergeschrieben werden, was man als *Regelbasis* bezeichnet. Eine solche Methode ist *C4.5Rules* (Quinlan 1993).

INTERPRETIERBARKEIT

IF-THEN-REGELN

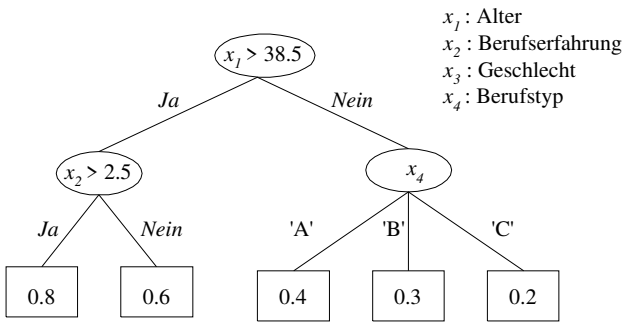


***Abb. 9.5*:** Beispiel für einen (hypothetischen) Entscheidungsbaum. Jeder Pfad von der Wurzel zu einem Blatt kann als eine konjunktive Regel niedergeschrieben werden, welche aus Bedingungen besteht, die sich durch die Entscheidungsknoten entlang des Pfades definieren.

Der Entscheidungsbaum aus Abbildung 9.5 zum Beispiel kann als die folgende Menge an Regeln aufgeschrieben werden:

R1: IF (Alter>38,5) AND (Berufserfahrung>2.5) THEN y =0,8
R2: IF (Alter>38,5) AND (Berufserfahrung≤2.5) THEN y =0,6
R3: IF (Alter≤38,5) AND (Berufstyp='A') THEN y =0,4
R4: IF (Alter≤38,5) AND (Berufstyp='B') THEN y =0,3
R5: IF (Alter≤38,5) AND (Berufstyp='C') THEN y =0,2 .

Wissensextraktion

Solch eine Regelbasis erlaubt die *Wissensextraktion*; sie ist leicht verständlich und erlaubt es Experten, das anhand der Daten erlernte Modell zu verifizieren. Für jede Regel kann ebenfalls der Prozentsatz an durch die Regel abgedeckten Daten ermittelt werden; dies ist der sogenannte *Support einer Regel*. Die Regeln spiegeln die Haupteigenschaften der Datenmenge wider. Sie zeigen wichtige Merkmale und Aufteilungsgrenzen auf. In unserem (hypothetischen) Szenario erkennen wir beispielsweise, dass sich hinsichtlich unserer Zielsetzung (y) Menschen der Altersgruppe von bis zu 38 Jahren von denen der Gruppe ab 39 Jahren aufwärts unterscheiden. Und in letzterer Gruppe ist es der Berufstyp, welcher sie unterscheidet, wohingegen es in ersterer Gruppe die Berufserfahrung ist, die als bestes Unterscheidungskriterium herangezogen werden kann.

Support einer Regel

Im Falle eines Klassifikationsbaumes kann es durchaus mehr als ein Blatt mit gleicher Klassenzuordnung geben. In diesem Fall können die multiplen konjunktiven Ausdrücke, welche den unterschiedlichen Pfaden entsprechen, als Disjunktion (OR) kombiniert werden. Die Klassenregion entspricht damit einer Vereinigung dieser zahlreichen einzelnen Teilregionen, wobei jede Teilregion für eine durch ein Blatt definierte Region steht. Klasse $\mathcal{C}_1$ aus Abbildung 9.1 schreibt man beispielsweise als

$$\text{IF } (x \leq w_{10}) \text{ OR } ((x_1 > w_{10}) \text{ AND } (x_2 \leq w_{20})) \text{ THEN } \mathcal{C}_1 \,.$$

Pruning von Regeln ist zur Vereinfachung möglich. Das Entfernen eines Unterbaums entspricht dem Pruning von Termen aus einer Anzahl von Regeln zur selben Zeit. Es ist durchaus möglich, einen Term aus einer Regel dem Pruning zu unterziehen, ohne andere Regeln einzubeziehen. Wenn wir in obiger Regelmenge für R3 zum Beispiel erkennen, dass alle, deren `Berufstyp=,,A‘‘` ist, Ergebnisse nahe 0,4 haben, und zwar unabhängig vom `Alter`, dann kann $R3$ wie folgt gekürzt werden:

Pruning von Regeln

$$R3' : \text{IF (Berufstyp='A') THEN } y = 0{,}4\,.$$

Man beachte, dass es nach dem Pruning der Regeln unter Umständen nicht mehr möglich ist, sie wieder in die Baumform zu bringen.

9.5 Lernen von Regeln anhand von Daten

Wie wir soeben gesehen haben, besteht ein Ansatz, um IF-THEN-Regeln zu erstellen, darin, einen Entscheidungsbaum zu testen und ihn in Regeln zu konvertieren. Eine andere Herangehensweise ist die, Regeln direkt zu lernen. Die *Regelinduktion* funktioniert ähnlich der Bauminduktion, abgesehen davon, dass die Regelinduktion eine Suche ist, die sich zuerst in die Tiefe bewegt und einen Pfad (eine Regel) pro Schritt generiert, wohingegen die Bauminduktion sich vornehmlich in die Breite bewegt und alle Pfade simultan generiert.

Regelinduktion

Regeln werden eine nach der anderen erlernt. Jede Regel ist eine Verkettung von Bedingungen für diskrete oder numerische Attribute (wie bei Entscheidungsbäumen) und diese Bedingungen werden eine nach der anderen hinzugefügt, um ein Kriterium zu optimieren, beispielsweise um die Entropie zu minimieren. Man spricht davon, dass eine Regel ein Beispiel *abdeckt*, wenn das Beispiel alle Bedingungen der Regel erfüllt. Ist eine Regel einmal gebildet und eventuell gekürzt worden, wird sie zur Regelbasis hinzugefügt und alle durch die Regel abgedeckten Beispiele werden aus dem Datensatz entfernt, woraufhin der Prozess fortgesetzt wird, bis genügend Regeln hinzugefügt wurden. Dies bezeichnet man als *sequentielle Abdeckung*. Es gibt eine äußere Schleife, in der die Regeln eine nach der anderen zur Regelbasis hinzugefügt werden, und eine innere Schleife, in der die Bedingungen nacheinander der gegenwärtigen Regel zugefügt werden. Dieses Vorgehen ist greedy und kann kein optimales Ergebnis garantieren. Beide Schleifen haben einen Pruning-Schritt für bessere Generalisierung.

sequentielle Abdeckung

Ein Beispiel für einen Regelinduktionsalgorithmus ist *Ripper* (Coehen 1995), welcher auf dem älteren Algorithmus *Irep* (Fürnkrantz und Widmer 1994) basiert. Wir beginnen mit dem Fall von zwei Klassen, bei dem wir von positiven und negativen Beispielen sprechen, und verallgemeinern später auf $K > 2$ Klassen. Regeln werden hinzugefügt, um

Ripper
Irep

positive Beispiele zu erklären, so dass, wenn eine Instanz durch keine Regel abgedeckt wird, sie als negativ klassifiziert wird. Wenn eine Regel zutrifft, handelt es sich entweder um eine korrekte Regel (wahr positiv) oder sie erzeugt ein falsches Positiv. Der Pseudocode der äußeren Schleife von Ripper findet sich im Listing 9.2.

FOIL-ALGORITHMUS

Bei Ripper werden Bedingungen zur Regel hinzugefügt, um ein Maß für den Informationszuwachs zu maximieren, welches in Quinlans (1990) *Foil-Algorithmus* Verwendung findet. Nehmen wir an, uns liegt Regel R vor und R' ist die in Frage kommende Regel, nachdem eine Bedingung hinzugefügt wurde. Die Zuwachsänderung definiert sich als

$$Zuwachs(R', R) = s \cdot \left(\log_2 \frac{N'_+}{N'} - \log_2 \frac{N_+}{N} \right) , \tag{9.17}$$

wobei N der Anzahl an durch R abgedeckten Instanzen entspricht und N_+ die Zahl an darin enthaltenen wahren Positiven darstellt. N' und N'_+ sind ähnlich definiert für R'. s ist die Anzahl an wahren Positiven in R, die immer noch wahre Positive in R' sind, nachdem die Bedingung hinzugefügt wurde. Im Kontext der Informationstheorie misst die Zuwachsänderung die Reduktion an Bits, um eine positive Instanz zu codieren.

REGELBEWERTUNGS-METRIK

Bedingungen werden zu einer Regel hinzugefügt, bis sie keine negativen Beispiele mehr abdeckt. Ist eine Regel einmal erstellt, wird sie dann zurückgekürzt, indem Bedingungen in umkehrter Reihenfolge gelöscht werden, um dadurch die Regel zu finden, welche die *Regelbewertungsmetrik* maximiert:

$$rvm(R) = \frac{p - n}{p + n} , \tag{9.18}$$

wobei p und n die Anzahl an wahren beziehungsweise falschen Positiven an der Pruning-Menge darstellen, welche einem Drittel der Daten entspricht, da zwei Drittel für die Menge zur Erstellung der Regeln genutzt wurden.

Ist eine Regel einmal erstellt und gekürzt, so werden alle positiven und negativen Beispiele, die durch die Regel abgedeckt werden, aus dem Datensatz entfernt. Wenn noch positive Beispiele übrig bleiben, wird die Regelinduktion fortgesetzt. Bei auftretendem Rauschen brechen wir unter Umständen vorzeitig ab, und zwar dann, wenn eine Regel nicht genügend Beispiele erklärt. Um den Wert einer Regel zu messen, nutzt man die minimale Beschreibungslänge (Abschnitt 4.8) (Quinlan 1995). Typischerweise brechen wir ab, wenn die Beschreibung einer Regel nicht kürzer ist als die Beschreibung der Instanzen, die sie erklärt. Die Beschreibungslänge einer Regelbasis ist die Summe der Beschreibungslängen aller Regeln in der Regelbasis plus die Beschreibung der Instanzen, die nicht durch die Regelbasis abgedeckt sind. Der Ripper-Algorithmus beendet das Hinzufügen von Regeln, wenn die Beschreibungslänge der Regelbasis mehr als 64 Bit länger ist als die bisherige beste Beschreibungslänge.

```
Ripper(Pos,Neg,k)
    RegelMenge ← LerneRegelMenge(Pos,Neg)
    For k Mal
        RegelMenge ← OptimiereRegelMenge(RegelMenge,Pos,Neg)
LerneRegelMenge(Pos,Neg)
    RegelMenge ← ∅
    DL ← BeschrLän(RegelMenge,Pos,Neg)
    Repeat
        Regel ← LerneRegel(Pos,Neg)
        Füge Regel zu RegelMenge hinzu
        DL' ← BeschrLän(RegelMenge,Pos,Neg)
        If DL'>DL+64
          PruningDerRegelMenge(RegelMenge,Pos,Neg)
          Return RegelMenge
        If DL'<DL DL ← DL'
          Lösche Instanzen, die durch Regel abgedeckt sind
              aus Pos und Neg
    Until Pos = ∅
    Return RegelMenge
PruningDerRegelMenge(RegelMenge,Pos,Neg)
    For jede Regel ∈ RegelMenge in umgekehrter Reihenfolge
        DL ← BeschrLän(RegelMenge,Pos,Neg)
        DL' ← BeschrLän(RegelMenge-Regel,Pos,Neg)
        IF DL'<DL Lösche Regel aus RegelMenge
    Return RegelMenge
OptimiereRegelMenge(RegelMenge,Pos,Neg)
     For jede Regel ∈ RegelMenge
         DL0 ← BeschrLän(RegelMenge,Pos,Neg)
         DL1 ← BeschrLän(RegelMenge-Regel+
                 ErsetzeRegel(RegelMenge,Pos,Neg),Pos,Neg)
         DL2 ← BeschrLän(RegelMenge-Regel+
                 RevidiereRegel(RegelMenge,Regel,Pos,Neg),Pos,Neg)
         If DL1=min(DL0,DL1,DL2)
                 Lösche Regel aus RegelMenge und
                         füge hinzu
                           ErsetzeRegel(RegelMenge,Pos,Neg)
         Else If DL2=min(DL0,DL1,DL2)
                 Lösche Regel aus RegelMenge und
                         füge hinzu
                           RevidiereRegel(RegelMenge,Regel,Pos,Neg)
     Return RegelMenge
```

***Listing 9.2*:** Der Ripper-Algorithmus zum Erlernen von Regeln. Nur die äußere Schleife ist dargestellt; die innere Schleife ähnelt dem Hinzufügen von Knoten in einem Entscheidungsbaum.

Sobald die Regelbasis erlernt ist, durchlaufen wir die Regeln in umgekehrter Reihenfolge, um zu prüfen, ob sie entfernt werden können, ohne die Beschreibungslänge zu erhöhen.

Regeln in der Regelbasis werden auch optimiert, nachdem sie erlernt wurden. Ripper betrachtet zwei Alternativen für eine Regel: die eine, die sogenannte Ersetzungsregel, beginnt mit einer leeren Regel, wird dann aufgebaut und schließlich verkürzt. Die andere, die Revisionsregel, beginnt mit der Regel in ihrem aktuellen Zustand, wird dann aufgebaut und zuletzt verkürzt. Diese zwei Alternativen werden mit der Originalregel verglichen und die kürzeste dieser drei wird der Regelbasis hinzugefügt. Diese Optimierung der Regelbasis kann k Mal durchgeführt werden, typischerweise aber zweimal.

Wenn es $K > 2$ Klassen gibt, werden sie hinsichtlich ihrer a-priori-Wahrscheinlichkeiten geordnet, so dass $\mathcal{C}_1$ die niedrigste und $\mathcal{C}_K$ die höchste a-priori-Wahrscheinlichkeit aufweist. Dann wird eine Sequenz von Zweiklassenproblemen definiert, und zwar so, dass als erstes Instanzen der Klasse $\mathcal{C}_1$ als positive Beispiele genutzt werden und Instanzen aller anderen Klassen als negative Beispiele betrachtet werden. Sodann, nachdem $\mathcal{C}_1$ gelernt und alle zugehörigen Instanzen entfernt wurden, wird gelernt, wie $\mathcal{C}_2$ von $\mathcal{C}_3, \ldots, \mathcal{C}_K$ zu unterscheiden ist. Dieser Prozess wird wiederholt, bis nur $\mathcal{C}_K$ übrig bleibt. Die leere Standardregel wird dann mit dem Label $\mathcal{C}_K$ versehen, so dass jede Instanz, die nicht durch irgendeine Regel abgedeckt ist, der Klasse $\mathcal{C}_K$ zugewiesen wird.

Für einen Datensatz der Größe N ist Rippers Komplexität gleich $\mathcal{O}(N \log^2 N)$; es handelt sich um einen Algorithmus, der für sehr große Datensätze verwendet werden kann (Dietterich 1997). Die Regeln, welche wir erlernen, sind *propositionale Regeln*. Stärkere Ausdruckskraft besitzen die *Regeln erster Ordnung* mit Variablen in ihren Bedingungen, die man als *Prädikate* bezeichnet. Ein *Prädikat* ist eine Funktion, die in Abhängigkeit vom Wert ihres Arguments entweder wahr oder falsch zurück gibt. Prädikate erlauben es somit, Relationen zwischen den Werten von Attributen zu definieren, was durch Propositionen nicht erreicht werden kann (Mitchell 1997):

PROPOSITIONALE REGELN

REGELN ERSTER ORDNUNG

$$\mathsf{IF\ Vater}(y, x)\ \mathsf{AND\ Weiblich}(y)\ \mathsf{THEN\ Tochter}(x, y)\ .$$

Solche Regeln können als Programme in einer logischen Programmiersprache wie Prolog verstanden werden, und das Erlernen der Regeln anhand von Daten bezeichnet man als *induktives logisches Programmieren*. Ein Algorithmus dieser Art ist *Foil* (Quinlan 1990).

INDUKTIVES LOGISCHES PROGRAMMIEREN

Die Zuweisung eines Werts zu einer Variable nennt man *Bindung*. Eine Regel trifft zu, wenn es für den gegebenen Trainingsdatensatz eine Menge von Bindungen gibt, die die Regel erfüllen. Das Lernen von Regeln ersten Grades ist ähnlich dem Erlernen von propositionalen Regeln mit einer äußeren Schleife für das Hinzufügen von Regeln, einer inneren Schleife

BINDUNG

für die Ergänzung einer Regel mit Bedingungen sowie Beschneidungen am Ende einer jeden Schleife. Der Unterschied findet sich in der inneren Schleife, wo wir bei jedem Schritt ein hinzuzufügendes Prädikat (statt einer Proposition) betrachten und die Verbessung der Leistungskraft der Regel beobachten (Mitchell 1997). Um diese Leistungskraft einer Regel zu berechnen, betrachten wir alle möglichen Bindungen der Variablen, zählen die Anzahl an positiven und negativen Bindungen im Datensatz und nutzen beispielsweise Gleichung 9.17. Im Falle von Regeln ersten Grades liegen uns Prädikate statt Propositionen vor, die somit bereits definiert worden sein sollten, und der Datensatz ist eine Menge an Prädikaten, von denen wir wissen, dass sie wahr sind.

9.6 Multivariate Bäume

Im Fall eines univariaten Baums wird nur eine Eingabedimension für die Entscheidung einer Aufteilung genutzt. Beim *multivariaten Baum* können an einem Entscheidungsknoten alle Eingabedimensionen verwendet werden; er ist somit allgemeiner. Wenn alle Eingaben numerisch sind, wird ein binärer, linearer, multivariater Knoten definiert als

MULTIVARIATER BAUM

$$f_m(\boldsymbol{x}) : \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0 \,. \tag{9.19}$$

Da der lineare multivariate Knoten eine gewichtete Summe berechnet, sollten diskrete Attribute durch 0/1-Dummy-Variablen repräsentiert werden. Gleichung 9.19 definiert eine Hyperebene mit willkürlicher Ausrichtung (siehe Abbildung 9.6). Sukzessive Knoten entlang eines Pfads von der Wurzel zu einem Blatt zerteilen diese weiter und Blattknoten definieren Polyeder im Eingaberaum. Der univariate Knoten mit einem numerischen Merkmal bildet einen Sonderfall, wenn alle w_{mj} bis auf eins gleich 0 sind. Somit definiert der univariate numerische Knoten aus Gleichung 9.1 auch eine lineare Diskriminante, jedoch eine, die orthogonal zur Achse x_j liegt, sie bei w_{m0} schneidet, und parallel zu allen anderen x_i verläuft. Wir sehen also, dass es an einem univariaten Knoten d mögliche Ausrichtungen ($\boldsymbol{w}_m$) gibt und $N_m - 1$ mögliche Schwellwerte ($-w_{m0}$), wodurch eine vollständige Suche ermöglicht wird. An einem multivariaten Knoten gibt es $2^d \binom{N_m}{d}$ mögliche Hyperebenen (Murthy, Kasif und Salzberg 1994) und eine vollständige Suche ist nicht mehr praktikabel.

Wenn wir von einem univariaten auf einen linearen multivariaten Knoten übergehen, wird der Knoten flexibler. Es ist sogar möglich, ihn noch flexibler zu gestalten, indem ein nichtlinearer multivariater Knoten genutzt wird. Im quadratischen Fall erhalten wir beispielsweise

$$f_m(\boldsymbol{x}) : \boldsymbol{x}^T \mathbf{W}_m \boldsymbol{x} + \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0 \,. \tag{9.20}$$

Guo und Gelfand (1992) schlagen die Nutzung eines mehrlagigen Perzeptrons (Kapitel 11) vor, welches eine lineare Summe von nichtlinearen

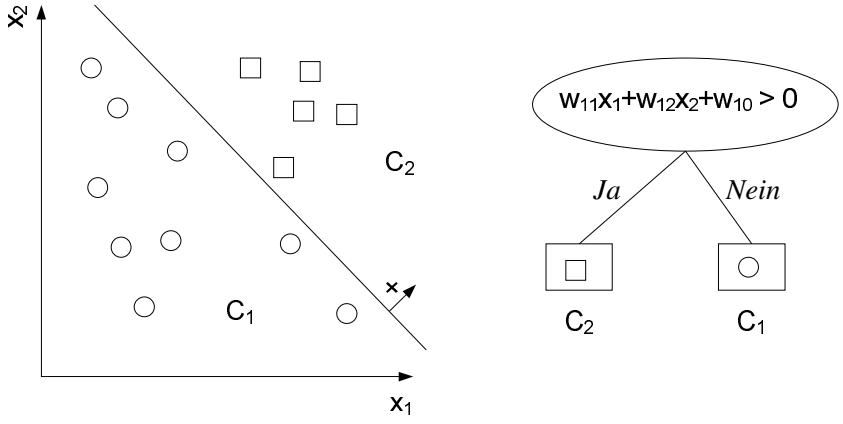


***Abb. 9.6*:** Beispiel für einen linearen multivariaten Entscheidungsbaum. Der lineare multivariate Knoten kann eine willkürliche Hyperebene platzieren und ist somit allgemeiner, wohingegen der univariate Knoten auf axial ausgerichtete Aufteilungen beschränkt ist.

Basisfunktionen ist und eine weitere Möglichkeit für die Nutzung von nichtlinearen Entscheidungsknoten darstellt. Noch eine andere Möglich-

SPHÄRENKNOTEN keit bietet ein *Sphärenknoten* (Devroye, Györfi und Lugosi 1996)

$$f_m(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{c}_m\| \leq \alpha_m \,, \tag{9.21}$$

wobei $\boldsymbol{c}_m$ dem Mittelpunkt und α_m dem Radius entspricht.

Es existieren eine Vielzahl von Algorithmen, die für das Lernen von multivariaten Entscheidungsbäumen für die Klassifikation aufgestellt wurden. Der älteste ist die multivariate Version des *CART-Algorithmus* (Breiman et al. 1984), welcher die schrittweise Feinabstimmung der Gewichte w_{mj} vornimmt, um die Unreinheit zu verringern. CART besitzt auch eine Vorverarbeitungsstufe, um die Dimensionalität mittels Teilmengenselektion (Kapitel 6) zu verringern und die Komplexität des Knotens zu reduzieren. Ein Algorithmus mit einigen Erweiterungen von CART

OC1 ist der *OC1-Algorithmus* (Murthy, Kasif, und Salzberg 1994). Eine weitere Möglichkeit (Loh und Vanichsetakul 1988) besteht darin, davon auszugehen, dass alle Klassen gaußverteilt mit einer gemeinsamen Kovarianzmatrix sind und dadurch lineare Diskriminanten besitzen, die eine Klasse von den anderen abtrennen (Kapitel 5). In solch einem Fall und bei K Klassen hat jeder Knoten K Verzweigungen, und jede Verzweigung entspricht einer Diskriminante, welche die eine Klasse von den anderen trennt. Brodley und Utgoff (1995) entwickelten eine Methode, bei der die linearen Diskriminanten trainiert werden, um den Klassifikationsfehler zu minimieren (Kapitel 10). Guo und Gelfand (1992) stellen eine Heuristik vor, um $K > 2$ Klassen in zwei Supergruppen aufzuteilen, woraufhin dann binäre multivariate Bäume gelernt werden können. Loh und Shih

(1997) nutzen die 2-Means-Clustermethode (Kapitel 7), um Daten in zwei Teile zu gruppieren. Yıldız und Alpaydın (2000) setzen die LDA (Kapitel 6) ein, um die Diskriminante zu finden, sobald die Klassen in zwei Gruppen aufgeteilt wurden.

Ein beliebiger Klassifikator approximiert die reale (unbekannte) Diskriminante, indem eine Hypothese aus der zugehörigen Hypothesenklasse gewählt wird. Wenn wir univariate Knoten nutzen, verwendet unsere Approximation stückweise axial ausgerichtete Hyperebenen. Bei den linearen multivariaten Knoten können wir willkürliche Hyperebenen nutzen und eine bessere Approximation unter Einsatz weniger Knoten erzielen. Wenn die zugrunde liegende Diskriminante gekrümmt ist, funktionieren die nichtlinearen Knoten besser. Der Verzweigungsfaktor hat einen ähnlichen Effekt dahingehend, dass er die Anzahl an Diskriminanten festlegt, welche durch einen Knoten definiert werden. Ein binärer Entscheidungsknoten mit zwei Verzweigungen definiert eine Diskriminante, welche den Eingaberaum zweiteilt. Ein n-Wegeknoten trennt in n Segmente. Somit besteht eine Abhängigkeit zwischen der Komplexität eines Knotens, dem Verzweigungsfaktor und der Baumgröße. Bei einfachen Knoten und niedrigen Verzweigungsfaktoren erstellt man zwar möglicherweise große Bäume, andererseits sind solche Bäume aber beispielsweise mit univariaten binären Knoten leichter zu interpretieren. Komplexere Knoten erfordern auch mehr Daten und sind anfällig für Überanpassung, je mehr wir in die Tiefe des Baums gelangen und je weniger Daten uns zur Verfügung stehen. Wenn die Knoten komplex sind und der Baum klein ist, so entfernen wir uns auch vom Grundkonzept des Baumes, nämlich dem, ein Problem in eine Menge von einfachen Problemen zu unterteilen. Letztendlich könnten wir einen sehr komplexen Klassifikator in der Wurzel nutzen, der alle Klassen voneinander abgrenzt – von einem Baum kann dann jedoch keine Rede mehr sein!

9.7 Anmerkungen

Die Strategie des Teilens und Beherrschens ist eine häufig genutzte Heuristik, die schon zu Cäsars Zeiten Verwendung gefunden hat, um komplexe Probleme, beispielsweise Gallien, in eine Gruppe von einfacheren Problemen herunterzubrechen. Bäume werden in der Informatik oft genutzt, um die Komplexität von linearer auf logarithmische Rechenzeit zu verringern. Entscheidungsbäume gewannen in der Statistik bei Breiman et al. (1984) an Popularität und im maschinellen Lernen bei Quinlan (1986, 1993). Multivariate Bauminduktionsmethoden erfreuen sich erst seit kurzem wachsenden Interesses; eine Betrachtung und ein Vergleich für viele Datensätze findet sich bei Yıldız und Alpaydın (2000). Diverse Forscher (z.B. Guo und Galfand 1992) haben vorgeschlagen, die Einfachheit von Bäumen mit der Genauigkeit von mehrlagigen Perzeptronen (Kapitel 11) zu kombinieren. Viele Studien kamen jedoch zu dem Schluss, dass die

univariaten Bäume recht akkurat und interpretierbar sind, und dass die zusätzliche Komplexität, die durch lineare (oder nichtlineare) multivariate Knoten verursacht werden würde, kaum gerechtfertigt sei. Eine aktuelle Übersicht ist in Rokach und Maimon (2005) zu finden.

OMNIVARIATER ENTSCHEIDUNGSBAUM

Der *omnivariate Entscheidungsbaum* (Yıldız und Alpaydın 2001) ist eine hybride Baumarchitektur, bei welcher der Baum entweder univariate, lineare multivariate oder nichtlineare multivariate Knoten haben kann. Die Idee ist folgende: jeder Entscheidungsknoten entspricht jeweils einem anderen Unterproblem, welches durch die Teilmenge der den Knoten erreichenden Daten definiert ist; somit kann unter Umständen während der Konstruktion ein jeweils anderes Modell als angemessen erscheinen, und daher sollte das angemessenste gefunden und verwendet werden. Überall denselben Knotentyp zu nutzen, entspricht der Annahme, dass die gleiche induktive Verzerrung in allen Teilen des Eingaberaums aufzufinden ist. Bei einem omnivariaten Baum werden an jedem Knoten in Frage kommende Knoten unterschiedlichen Typs ausprobiert und mit Hilfe eines statistischen Tests (Kapitel 19) an einem Validierungsdatensatz miteinander verglichen, um zu bestimmen, welcher am besten generalisiert. Der einfachere Knoten wird ausgewählt, es sei denn, es kann bewiesen werden, dass ein komplexerer signifikant höhere Genauigkeit liefert. Die Ergebnisse zeigen, dass komplexere Knoten frühzeitig im Baum genutzt werden, also näher an der Wurzel, und dass mit zunehmender Tiefe am Baum auch einfache univariate Knoten genügen. Wenn wir uns den Blättern nähern, liegen einfachere Probleme, aber auch weniger Daten vor. In solch einem Fall verursachen komplexe Knoten eine Überanpassung und werden durch den statistischen Test abgewiesen. Die Anzahl an Knoten erhöht sich exponential, je mehr wir im Baum in die Tiefe gehen; daher sind eine Mehrzahl der Knoten univariat und die insgesamte Komplexität erhöht sich kaum.

Entscheidungsbäume werden öfter für die Klassifikation als für die Regression verwendet. Sie sind sehr beliebt. Sie lernen und reagieren schnell und sind außerdem in vielen Bereichen akkurat (Murthy 1998). Es gibt sogar den Fall, dass ein Entscheidungsbaum genaueren Methoden aufgrund seiner Interpretierbarkeit vorgezogen wird. Wenn er in Form einer Menge von IF-THEN-Regeln vorliegt, ist der Baum verständlich und die Regeln können von menschlichen Experten mit Kenntnis des Anwendungsbereichs validiert werden.

Es wird im Allgemeinen empfohlen, dass ein Entscheidungsbaum getestet und seine Genauigkeit als Maßstab festgehalten wird, bevor kompliziertere Algorithmen angewandt werden. Die Analyse des Baums erlaubt uns außerdem, seine wichtigen Merkmale zu verstehen; der univariate Baum nimmt seine eigene Merkmalsextraktion vor. Ein weiterer großer Vorteil des univariaten Baums besteht darin, dass er numerische und diskrete Merkmale zusammen nutzen kann, ohne dass der eine Typ in den anderen konvertiert werden muss.

Der Entscheidungsbaum ist eine nichtparametrische Methode, ähnlich den in Kapitel 8 besprochenen Methoden, jedoch bestehen diverse Unterschiede:

- Jeder Blattknoten entspricht einem „Intervall“, allerdings müssen die Intervalle nicht dieselbe Größe haben (wie bei Parzen-Fenstern) oder dieselbe Anzahl an Instanzen enthalten (wie bei k-Nächste-Nachbarn-Methoden).
- Die Intervallaufteilungen werden nicht nur basierend auf Ähnlichkeit im Eingaberaum vorgenommen, sondern auch die geforderte Ausgabeinformation wird mittels Entropie oder des mittleren quadratischen Fehlers verwendet.
- Ein weiterer Vorteil von Entscheidungsbäumen ist der, dass das Blatt („das Intervall“) aufgrund der Baumstruktur viel schneller mit einer geringeren Anzahl an Vergleichen gefunden wird.
- Ist der Entscheidungsbaum einmal erstellt, speichert er nicht den gesamten Datensatz ab, sondern nur die Struktur des Baums, die Parameter der Entscheidungsknoten und die Ausgabewerte in den Blättern. Dies impliziert, dass die Speicherkomplexität ebenfalls niedriger ist, im Gegensatz zu instanzbasierten nichtparametrischen Methoden, die alle Beispiele speichern.

Bei einem Entscheidungsbaum muss eine Klasse keine einzelne Beschreibung besitzen, mit der alle Instanzen übereinstimmen sollten. Sie kann mehrere mögliche Beschreibungen haben, die im Eingaberaum sogar disjunkt sein können.

Die bisher diskutierten Entscheidungsbäume haben *harte* Entscheidungsknoten, d. h., wir nehmen einen der Zweige in Abhängigkeit vom Test. Wir beginnen bei der Wurzel, folgen einem einzelnen Pfad und stoppen bei einem Blatt, wo wir den dort gespeicherten Antwortwert ausgeben. Bei einem *weichen Entscheidungsbaum* dagegen nehmen wir *alle* Zweige, allerdings mit unterschiedlichen Wahrscheinlichkeiten; wir folgen parallel allen Pfaden und erreichen die Blätter, jedoch mit unterschiedlichen Wahrscheinlichkeiten. Die Ausgabe ist das gewichtete Mittel aus allen Ausgaben in den verschiedenen Blättern, wobei die Gewichte den über die Pfade akkumulierten Wahrscheinlichkeiten entsprechen. Wir werden dies in Abschnitt 12.9 diskutieren.

WEICHER ENTSCHEIDUNGSBAUM

In Kapitel 17 werden wir uns mit dem Kombinieren mehrerer Lerner beschäftigen. Eines der populärsten Modelle, das dabei verwendet wird, ist der Entscheidungsbaum, und ein Ensemble von Entscheidungsbäumen wird als *Entscheidungswald* bezeichnet. Wenn wir nicht nur einen, sondern viele Entscheidungsbäume trainieren – jeden davon auf einer zufälligen Teilmenge der Trainingsmenge oder einer zufälligen Teilmenge der Eingabemerkmale – und ihre Vorhersagen kombinieren, dann werden

ENTSCHEIDUNGSWALD

wir feststellen, dass die Gesamtgenauigkeit signifikant gesteigert werden kann. Das ist die Idee, die hinter dem *Zufallswald*-Verfahren steht.

ZUFALLSWALD

Der Baum unterscheidet sich von den statistischen Modellen, die wir in den vergangen Kapiteln besprochen haben. Der Baum codiert die Diskriminanten direkt, welche die Klasseninstanzen voneinander trennen, ohne dabei der Frage viel Beachtung zu schenken, wie diese Instanzen in den Regionen verteilt sind. Der Entscheidungsbaum ist eine *diskriminantenbasierte* Methode, wohingegen die statistischen Methoden auf der *Likelihood basieren*, und zwar dahingehend, dass sie $p(\boldsymbol{x}|\mathcal{C}_i)$ explizit schätzen, bevor sie den Satz von Bayes anwenden und die Diskriminante berechnen. Diskriminantenbasierte Methoden schätzen die Diskriminante direkt und umgehen die Schätzung der Klassendichten. In den vor uns liegenden Kapiteln werden wir diskriminantenbasierte Methoden weiter behandeln.

9.8 Übungen

1. Verallgemeinern Sie den Gini-Index (Gleichung 9.5) und den Fehlklassifikationsfehler (Gleichung 9.6) für $K > 2$ Klassen. Generalisieren Sie den Fehlklassifikationsfehler für das Risiko unter Einbeziehung einer Verlustfunktion.

   LÖSUNG:

   - Gini-Index mit $K > 2$ Klassen:
     $\phi(p_1, p_2, \ldots, p_K) = \sum_{i=1}^{K} \sum_{j<i} 2 p_i p_j$
   - Fehlklassifikationsfehler: $\phi(p_1, p_2, \ldots, p_K) = 1 - \max_{i=1}^{K} p_i$
   - Risiko: $\phi_\Lambda(p_1, p_2, \ldots, p_K) = \min_{i=1}^{K} \sum_{k=1}^{K} \lambda_{ik} p_k$, wobei Λ die $K \times K$-Matrix ist.

2. Für eine numerische Eingabe kann man statt einer binären Aufteilung eine dreifache Aufteilung mit zwei Schwellwerten und drei Verzweigungen nutzen mit

   $$x_j < w_{ma},\ w_{ma} \leq x_j < w_{mb},\ x_j \geq w_{mb}\ .$$

   Erstellen Sie eine Modifikation der Bauminduktionsmethode, um die zwei Schwellwerte w_{ma}, w_{mb} zu lernen. Worin liegen die Vorteile und die Nachteile solch eines Knotens gegenüber einem binären Knoten?

   LÖSUNG: Für die numerischen Attribute müssen wir alle möglichen Paare von Aufteilungsschwellwerten testen und das beste wählen. Wenn es zwei Aufteilungen gibt, dann gibt es drei Kinder, und bei der Berechnung der Entropie nach der Aufteilung müssen wir

über die drei Mengen summieren, die den Instanzen der drei Zweige entsprechen.

Die Komplexität des Problems, das beste Paar zu finden, ist $\mathcal{O}(\mathcal{N}_m^2)$ anstatt $\mathcal{O}(\mathcal{N}_m)$, und jeder Knoten speichert zwei Schwellwerte anstatt einen und hat drei Zweige anstatt zwei. Der Vorteil ist, dass ein ternärer Knoten eine Eingabe in drei Teile teilt, während dies zwei aufeinanderfolgende binäre Knoten erfordert. Welcher besser ist, hängt von den vorliegenden Daten ab. Wenn unsere Hypothesen beschränkte Intervalle erfordern (zum Beispiel Rechtecke), dann kann ein ternärer Knoten vorteilhaft sein.

3. Schreiben Sie einen Bauminduktionsalgorithmus mit Zurückverfolgung des Pfads (backtracking).

4. Bei der Generierung eines univariaten Baums kann ein diskretes Attribut mit n möglichen Werten durch n 0/1-Dummy-Variablen repräsentiert und dann wie n separate numerische Attribute behandelt werden. Was sind hier die Vor- und Nachteile dieser Vorgehensweise?

5. Leiten Sie einen Lernalgorithmus für Sphärenbäume ab (Gleichung 9.21). Verallgemeinern Sie auf Ellipsoidenbäume.

6. Für einem Regressionsbaum haben wir besprochen, dass an einem Blattknoten statt der Berechnung des Mittelwerts eine lineare Regressionsanpassung vorgenommen und eine Abhängigkeit zwischen der Antwort am Blatt und der Eingabe hergestellt werden kann. Entwickeln Sie eine ähnliche Methode für Klassifikationsbäume.

   Lösung: Das bedeutet, dass wir an jedem Blatt einen linearen Klassifikator haben, der mit den dort hinreichenden Instanzen trainiert wird. Dieser lineare Klassifikator erzeugt a-posteriori-Wahrscheinlichkeiten für die verschiedenen Klassen, und diese Wahrscheinlichkeiten werden bei der Berechnung der Entropie benutzt. Das heißt, es ist nicht nötig, dass ein Blatt rein ist (also nur Instanzen einer einzigen Klasse enthält); es genügt, dass der Klassifikator in diesem Blatt a-posteriori-Wahrscheinlichkeiten nahe 0 oder 1 erzeugt.

7. Schlagen Sie einen Regelinduktionsalgorithmus für die Regression vor.

8. Wie können wir bei Regressionsbäumen Unstetigkeiten an den Blattgrenzen loswerden?

9. Angenommen, wir haben für ein Klassifikationsproblem bereits einen trainierten Entscheidungsbaum. Wie können wir diesen zusätzlich zur Trainingsmenge dafür nutzen, einen k-NN-Klassifikator zu konstruieren?

   Lösung: Der Entscheidungsbaum macht eine Merkmalsselektion, und wir können nur die Merkmale verwenden, die von dem Baum

verwendet werden. Die mittlere Anzahl der Instanzen pro Blatt liefert uns auch die Information über einen guten k-Wert.

10. In einem multivariaten Baum werden wir in jedem Knoten mit großer Wahrscheinlichkeit nicht alle Eingabevariablen benötigen. Wie können wir die Dimensionalität in einem Knoten reduzieren?

    LÖSUNG: Jeder Teilbaum behandelt ein lokales Gebiet im Eingaberaum, das durch eine kleine Anzahl von Merkmalen erklärt werden kann. Wir können eine Merkmalsselektion oder -extraktion unter alleiniger Verwendung der Instanzen durchführen, die diesen Knoten erreichen. Idealerweise erwarten wir, dass wir während des Absteigens im Baum immer weniger Merkmale brauchen.

9.9 Literaturangaben

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Belmont, CA: Wadsworth International Group.

Brodley, C. E., and P. E. Utgoff. 1995. „Multivariate Decision Trees.“ *Machine Learning* 19: 45–77.

Cohen, W. 1995. „Fast Effective Rule Induction.“ In *Twelfth International Conference on Machine Learning,* ed. A. Prieditis and S. J. Russell, 115–123. San Mateo, CA: Morgan Kaufmann.

Devroye, L., L. Györfi, and G. Lugosi. 1996. *A Probabilistic Theory of Pattern Recognition.* New York: Springer.

Dietterich, T. G. 1997. „Machine Learning Research: Four Current Directions.“ *AI Magazine* 18: 97–136.

Fürnkrantz, J., and G. Widmer. 1994. „Incremental Reduced Error Pruning.“ In *Eleventh International Conference on Machine Learning,* ed. W. Cohen and H. Hirsh, 70–77. San Mateo, CA: Morgan Kaufmann.

Guo, H., and S. B. Gelfand. 1992. „Classification Trees with Neural Network Feature Extraction.“ *IEEE Transactions on Neural Networks* 3: 923–933.

Loh, W.-Y., and Y. S. Shih. 1997. „Split Selection Methods for Classification Trees.“ *Statistica Sinica* 7: 815–840.

Loh, W.-Y., and N. Vanichsetakul. 1988. „Tree-Structured Classification via Generalized Discriminant Analysis.“ *Journal of the American Statistical Association* 83: 715–725.

Mitchell, T. 1997. *Machine Learning.* New York: McGraw-Hill.

Murthy, S. K. 1998. „Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey.“ *Data Mining and Knowledge Discovery* 4: 345–389.

Murthy, S. K., S. Kasif, and S. Salzberg. 1994. „A System for Induction of Oblique Decision Trees.“ *Journal of Artificial Intelligence Research* 2: 1–32.

Quinlan, J. R. 1986. „Induction of Decision Trees.“ *Machine Learning* 1: 81–106.

Quinlan, J. R. 1990. „Learning Logical Definitions from Relations.“ *Machine Learning* 5: 239–266.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. 1995. „MDL and Categorical Theories (continued).“ In *Twelfth International Conference on Machine Learning,* ed. A. Prieditis and S. J. Russell, 467–470. San Mateo, CA: Morgan Kaufmann.

Rokach, L., and O. Maimon. 2005. „Top-Down Induction of Decision Trees Classifiers – A Survey.“ *IEEE Transactions on Systems, Man, and Cybernetics. Part C* 35: 476–487.

Yıldız, O. T., and E. Alpaydın. 2000. „Linear Discriminant Trees.“ In *Seventeenth International Conference on Machine Learning,* ed. P. Langley, 1175–1182. San Francisco: Morgan Kaufmann.

Yıldız, O. T., and E. Alpaydın. 2001. „Omnivariate Decision Trees.“ *IEEE Transactions on Neural Networks* 12: 1539–1546.

10 Lineare Diskriminanz

Bei der linearen Diskriminanz nehmen wir an, dass Instanzen einer Klasse auf lineare Weise von Instanzen anderer Klassen getrennt werden können. Hierbei handelt es sich um einen diskriminanzbasierten Ansatz, welcher die Parameter der linearen Diskriminanz direkt aus einer gegebenen zugeordneten Stichprobe schätzt.

10.1 Einführung

Wir erinnern aus den vorhergegangenen Kapiteln, dass wir bei der Klassifikation eine Menge von Diskriminanzfunktionen, $g_j(\boldsymbol{x})$ mit $j = 1, \ldots, K$, definieren und dann wie folgt vorgehen:

$$\text{wähle } \mathcal{C}_i, \text{ falls } g_i(\boldsymbol{x}) = \max_{j=1}^{K} g_j(\boldsymbol{x}) \; .$$

Bisher haben wir bei der Diskussion von Methoden für die Klassifikation zuerst die a-priori-Wahrscheinlichkeiten $\hat{P}(\mathcal{C}_i)$ und die Likelihoods der Klassen, $\hat{p}(\boldsymbol{x}|\mathcal{C}_i)$, geschätzt und danach den Satz von Bayes genutzt, um die a-posteriori-Dichten zu berechnen. Sodann definierten wir die Diskriminanzfunktionen hinsichtlich der a-posteriori-Wahrscheinlichkeiten, zum Beispiel

$$g_i(\boldsymbol{x}) = \log \hat{P}(\mathcal{C}_i|\boldsymbol{x}) \; .$$

LIKELIHOOD-BASIERTE KLASSIFIKATION

Dies bezeichnet man als *Likelihood-basierte Klassifikation*, und wir haben bereits die parametrischen (Kapitel 5), die semiparametrischen (Kapitel 7) und die nichtparametrischen Ansätze (Kapitel 8) zur Schätzung der Klassen-Likelihoods, $p(\boldsymbol{x}|\mathcal{C}_i)$, diskutiert.

DISKRIMINANZ-BASIERTE KLASSIFIKATION

Nun werden wir uns mit der *diskriminanzbasierten Klassifikation* befassen, bei der wir direkt für die Diskriminanzfunktion ein Modell vermuten, und dabei die Schätzung der Likelihoods oder a-posteriori-Wahrscheinlichkeiten umgehen. Wir wir bereits in Kapitel 9 für Entscheidungsbäume gesehen haben, stellt der diskriminanzbasierte Ansatz eine Vermutung über die Form der Diskriminanzfunktion zwischen den Klassen an, mutmaßt aber nicht und verlangt auch keinerlei Kenntnis über die Dichten, beispielsweise, ob sie gaußverteilt sind oder, ob die Eingaben korrelieren.

Wir definieren ein Modell für die Diskriminanzfunktion

$$g_i(\boldsymbol{x}|\Phi_i)$$

und parametrisieren es explizit mit der Parametermenge Φ_i; bei einem auf der Likelihood basierenden Schema würden wir stattdessen implizit Parameter bei der Definition der Likelihoodverteilungen nutzen. Es handelt sich um eine andere Art der induktiven Verzerrung: anstatt eine Vermutung hinsichtlich der Form der Dichten anzustellen, mutmaßen wir hinsichtlich der Form der Grenzen, welche die Klassen separieren.

Das Lernen ist dabei die Optimierung der Modellparameter Φ_i, um die Qualität der Separation, d. h. die Klassifikationsgenauigkeit an einem gegebenen Datensatz mit zugeordneten Instanzen zu maximieren. Dies unterscheidet sich von den Likelihood-Methoden, die separat für jede Klasse nach den Parametern suchen, welche die Likelihoods der Stichprobe maximieren.

Beim diskriminanzbasierten Ansatz interessiert uns nicht die korrekte Schätzung der Dichten innerhalb von Klassenregionen; das einzige was für uns von Interesse ist, ist die korrekte Schätzung der *Grenzen* zwischen den Klassenregionen. Befürworter des diskriminanzbasierten Ansatzes (z. B. Vapnik 1995) behaupten, dass die Schätzung der Klassendichten eine schwierigere Aufgabe sei als die Schätzung der Klassendiskriminanzen, und dass es keinen Sinn ergäbe, ein schweres Problem zu lösen, nur um dadurch ein leichteres Problem lösen zu können. Das ist natürlich nur dann wahr, wenn die Diskriminanz durch eine einfache Funktion approximiert werden kann.

In diesem Kapitel befassen wir uns mit dem einfachsten Fall, bei dem die Diskriminanzfunktionen linear in $\boldsymbol{x}$ sind:

$$g_i(\boldsymbol{x}|\boldsymbol{w}_i, w_{i0}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} \; . \tag{10.1}$$

LINEARE DISKRIMINANZFUNKTION

Lineare Diskriminanzfunktionen werden oft verwendet, hauptsächlich aufgrund ihrer Einfachheit, denn die Komplexitäten in Raum und Zeit sind $\mathcal{O}(d)$. Das lineare Modell ist einfach zu verstehen: die finale Ausgabe ist eine gewichtete Summe der Eingabeattribute x_j. Die Größenordnung des Gewichts w_j zeigt die Wichtigkeit von x_j an, und das Vorzeichen gibt Auskunft darüber, ob es sich um einen positiven oder negativen Effekt handelt. Die meisten Funktionen sind dahingehend additiv, als dass die Ausgabe der Summe der Effekte mehrerer Attribute entspricht, wobei die Gewichte positiv (verstärkend) oder negativ (abschwächend) sein können. Zum Beispiel berechnen Finanzinstitute für einen Darlehensantrag eines Kunden den sogenannten „Credit Score“, also die Kreditwürdigkeit dieses Kunden, die im allgemeinen als Summe der Effekte diverser Attribute ausgedrückt wird; beispielsweise hat das jährliche Einkommen hier einen positiven Effekt (höheres Einkommen erhöht die Kreditwürdigkeit).

Bei vielen Anwendungen ist die lineare Diskriminanzfunktion auch recht genau. Wir wissen zum Beispiel, dass bei gaußverteilten Klassen mit einer gemeinsamen Kovarianzmatrix die optimale Diskriminanzfunktion linear ist. Die lineare Diskriminanzfunktion kann jedoch auch dann genutzt werden, wenn diese Annahme nicht haltbar ist, und die Modellparameter können berechnet werden, ohne jegliche Mutmaßungen hinsichtlich der Klassendichten anzustellen. Wir sollten immer die lineare Diskriminanzfunktion nutzen, bevor wir ein komplizierteres Modell ausprobieren, um dadurch sicherzustellen, dass die zusätzliche Komplexität gerechtfertigt ist.

Wie immer formulieren wir das Problem, eine lineare Diskriminanzfunktion zu finden, als eine Suche nach den Parameterwerten, durch die eine Fehlerfunktion minimiert wird. Speziell konzentrieren wir uns auf *Gradientenmethoden* für die Optimierung einer Kriteriumsfunktion.

10.2 Generalisierung des linearen Modells

Ist ein lineares Modell nicht flexibel genug, so können wir die *quadratische Diskriminanzfunktion* verwenden und die Komplexität erhöhen:

QUADRATISCHE DISKRIMINANZFUNKTION

$$g_i(\boldsymbol{x}|\mathbf{W}_i, \boldsymbol{w}_i, w_{i0}) = \boldsymbol{x}^T\mathbf{W}_i\boldsymbol{x} + \boldsymbol{w}_i\boldsymbol{x} + w_{i0}\,. \tag{10.2}$$

Dieser Ansatz ergibt jedoch $O(d^2)$ und wir stoßen erneut auf das Dilemma von Verzerrung/Varianz: das quadratische Modell ist zwar allgemeiner, erfordert jedoch größere Datensätze und könnte unter Umständen zu einer Überanpassung bei kleinen Stichproben führen.

Ein äquivalenter Weg besteht darin, die Eingabe einer Vorbearbeitung zu unterziehen, indem *Terme höherer Ordnung*, auch bekannt als *Produktterme*, hinzugefügt werden. Beispielsweise können wir bei zwei Eingaben x_1 und x_2 neue Variablen

TERME HÖHERER ORDNUNG

PRODUKTTERME

$$z_1 = x_1, z_2 = x_2, z_3 = x_1^2, z_4 = x_2^2, z_5 = x_1x_2$$

definieren und $\boldsymbol{z} = [z_1, z_2, z_3, z_4, z_5]^T$ als Eingabe nutzen. Die im fünfdimensionalen $\boldsymbol{z}$-Raum definierte lineare Funktion entspricht einer nichtlinearen Funktion im zweidimensionalen $\boldsymbol{x}$-Raum. Statt eine nichtlineare Funktion (Diskriminanz- oder Regressionsfunktion) im Originalraum zu definieren, bestimmen wir stattdessen eine geeignete nichtlineare Transformation in einen neuen Raum, in dem die Funktion in linearer Form geschrieben werden kann.

Wir schreiben die Diskriminanzfunktion als

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{k} w_j\phi_{ij}(\boldsymbol{x})\,, \tag{10.3}$$

BASISFUNKTION

wobei $\phi_{ij}(\boldsymbol{x})$ *Basisfunktionen* sind. Terme höherer Ordnung sind nur ein möglicher Satz von Basisfunktionen; andere Beispiele sind

- $\sin(x_1)$
- $\exp(-(x_1 - m)^2/c)$
- $\exp(-\|\boldsymbol{x} - \boldsymbol{m}\|^2/c)$
- $\log(x_2)$
- $1(x_1 > c)$
- $1(ax_1 + bx_2 > c)$.

Dabei sind m, a, b, c Skalare, $\boldsymbol{m}$ ist ein d-dimensionaler Vektor und $1(b)$ liefert 1, wenn b wahr ist und 0, wenn b falsch ist. Die Idee, eine nichtlineare Funktion als lineare Summe von nichtlinearen Basisfunktionen zu schreiben, ist keine neue und war ursprünglich unter dem Begriff *Potenzialfunktionen* bekannt (Aizerman, Bravermans und Rozonoer 1964). Mehrlagige Perzeptronen (Kapitel 11) und radiale Basisfunktionen (Kapitel 12) haben den weiteren Vorteil, dass die Parameter der Basisfunktionen während des Lernens an die Daten feinangepasst werden können. In Kapitel 13 diskutieren wir Support-Vektor-Maschinen, die aus solchen Basisfunktionen konstruierte Kernel-Funktionen verwenden.

POTENZIAL-FUNKTION

10.3 Geometrie der linearen Diskriminanz

10.3.1 Zwei Klassen

Beginnen wir mit dem einfacheren Fall von zwei Klassen. In solch einem Fall ist eine Diskriminanzfunktion ausreichend:

$$\begin{aligned} g(\boldsymbol{x}) &= g_1(\boldsymbol{x}) - g_2(\boldsymbol{x}) \\ &= (\boldsymbol{w}_1^T\boldsymbol{x} + w_{10}) - (\boldsymbol{w}_2^T\boldsymbol{x} + w_{20}) \\ &= (\boldsymbol{w}_1 - \boldsymbol{w}_2)^T\boldsymbol{x} + (w_{10} - w_{20}) \\ &= \boldsymbol{w}^T\boldsymbol{x} + w_0 \,, \end{aligned}$$

und somit gilt

$$\text{wähle } \begin{cases} \mathcal{C}_1 \text{ falls } g(\boldsymbol{x}) > 0 \,, \\ \mathcal{C}_2 \text{ andernfalls} \,. \end{cases}$$

GEWICHTSVEKTOR
SCHWELLWERT

Hierdurch wird eine Hyperebene definiert, bei der $\boldsymbol{w}$ der *Gewichtsvektor* und w_0 der *Schwellwert* ist. Die letztere Bezeichnung stammt daher, dass

die Entscheidungsregel wie folgt umformuliert werden kann: Wähle $\mathcal{C}_1$, falls $\boldsymbol{w}^T\boldsymbol{x} > -w_0$; andernfalls wähle $\mathcal{C}_2$. Die Hyperebene zerlegt den Eingaberaum in zwei Halbräume: die Entscheidungsregion $\mathcal{R}_1$ für $\mathcal{C}_1$ und $\mathcal{R}_2$ für $\mathcal{C}_2$. Ein beliebiges $\boldsymbol{x}$ in $\mathcal{R}_1$ liegt auf der *positiven* Seite der Hyperebene und jedes $\boldsymbol{x}$ in $\mathcal{R}_2$ fällt in die *negative* Seite. Ist $\boldsymbol{x}$ gleich $\mathbf{0}$, dann $g(\boldsymbol{x}) = w_0$ und wir erkennen, dass bei $w_0 > 0$ der Ursprung auf der positiven Seite der Hyperebene liegt, bei $w_0 < 0$ findet er sich auf der negativen Seite, und falls $w_0 = 0$, so schneidet die Hyperebene den Ursprung (siehe Abbildung 10.1).

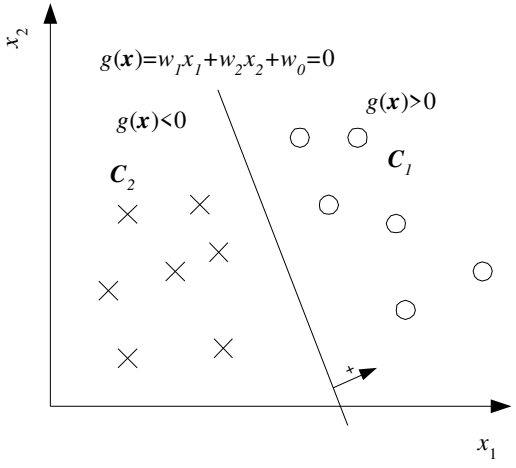


***Abb. 10.1*:** Im zweidimensionalen Fall ist die lineare Diskriminanzfunktion eine Linie, welche die Beispiele aus zwei Klassen voneinander trennt.

Nimmt man zwei Punkte $\boldsymbol{x}_1$ und $\boldsymbol{x}_2$, beide auf der Trennhyperebene, das heißt $g(\boldsymbol{x}_1) = g(\boldsymbol{x}_2) = 0$, dann ergibt sich

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_1 + w_0 &= \boldsymbol{w}^T\boldsymbol{x}_2 + w_0\,, \\ \boldsymbol{w}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) &= 0 \end{aligned}$$

und wir erkennen, dass $\boldsymbol{w}$ normal bezüglich beliebiger auf der Hyperebene liegender Vektoren ist. Schreiben wir $\boldsymbol{x}$ um als (Duda, Hart und Stork 2001)

$$\boldsymbol{x} = \boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\,,$$

wobei $\boldsymbol{x}_p$ die Normalprojektion von $\boldsymbol{x}$ auf die Hyperebene ist und r uns die Distanz von $\boldsymbol{x}$ zur Hyperebene angibt; diese ist negativ, wenn $\boldsymbol{x}$ auf der negativen Seite liegt und positiv, falls $\boldsymbol{x}$ auf der positiven Seite liegt (siehe Abbildung 10.2). Durch die Berechnung von $g(\boldsymbol{x})$ und wissend, dass $g(\boldsymbol{x}_p) = 0$, erhalten wir

$$r = \frac{g(\boldsymbol{x})}{\|\boldsymbol{w}\|}\,. \tag{10.4}$$

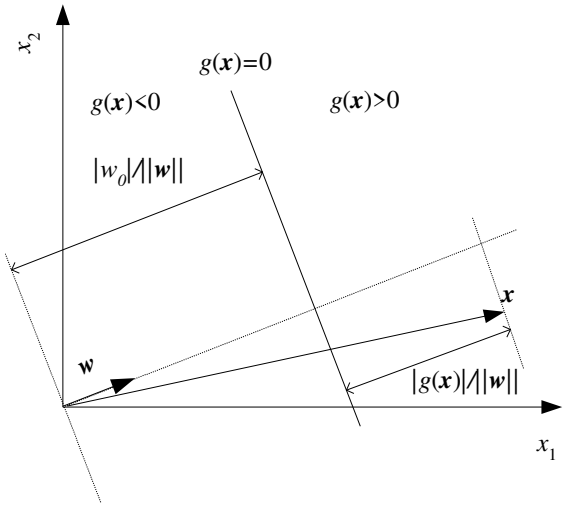


***Abb. 10.2*:** Geometrische Interpretation der linearen Diskriminanzfunktion.

Wir erkennen dann, dass die Distanz zum Ursprung sich wie folgt ergibt:

$$r_0 = \frac{w_0}{\|\boldsymbol{w}\|} \,. \tag{10.5}$$

Somit bestimmt w_0 die Lage der Hyperebene hinsichtlich des Ursprungs und $\boldsymbol{w}$ bestimmt ihre Ausrichtung.

10.3.2 Multiple Klassen

Wenn $K > 2$ Klassen vorliegen, so gibt es K Diskriminanzfunktionen. Sind sie linear, so erhalten wir

$$g_i(\boldsymbol{x}|\boldsymbol{w}_i, w_{i0}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} \,. \tag{10.6}$$

Wir werden uns später mit dem Lernprozess befassen, doch für den Augenblick nehmen wir an, dass die Parameter $\boldsymbol{w}_i, w_{i0}$ berechnet sind, so dass

$$g_i(\boldsymbol{x}|\boldsymbol{w}_i, w_{i0}) = \begin{cases} > 0 \text{ falls } \boldsymbol{x} \in \mathcal{C}_i \,, \\ \leq 0 \text{ andernfalls} \end{cases} \tag{10.7}$$

für alle $\boldsymbol{x}$ im Datensatz gilt. Die Nutzung solcher Diskriminanzfunktionen entspricht der Annahme, dass alle Klassen *linear trennbar* sind; das heißt, für jede Klasse $\mathcal{C}_i$ existiert eine Hyperebene H_i, und zwar so, dass alle $\boldsymbol{x} \in \mathcal{C}_i$ auf ihrer positiven Seite und alle $\boldsymbol{x} \in \mathcal{C}_j, j \neq i$ auf der negativen Seite liegen (siehe Abbildung 10.3).

LINEAR TRENNBARE KLASSEN

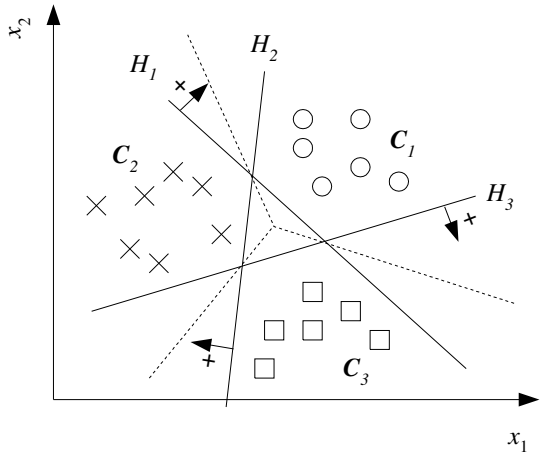


***Abb. 10.3*:** Bei der linearen Klassifikation trennt jede Hyperebene H_i die Beispiele aus $\mathcal{C}_i$ von denen aller anderen Klassen. Damit dies funktioniert, sollten die Klassen linear trennbar sein. Gepunktete Linien repräsentieren die induzierten Grenzen des linearen Klassifikators.

Während der Testphase und bei gegebenem $\boldsymbol{x}$ sollte im Idealfall nur ein $g_j(\boldsymbol{x}), j = 1, \ldots, K$ größer als 0 vorliegen und alle anderen sollten kleiner als 0 sein; dies ist jedoch nicht immer der Fall: die positiven Halbräume der Hyperebenen könnten sich überlappen, oder es könnte der Fall vorliegen, dass alle $g_j(\boldsymbol{x}) < 0$ sind. Diese könnten als *Ablehnungsfälle* eingestuft werden, doch das gewöhnliche Vorgehen besteht darin, $\boldsymbol{x}$ der Klasse zuzuweisen, welche die höchste Diskriminanz besitzt:

$$\text{Wähle } \mathcal{C}_i, \text{ falls } g_i(\boldsymbol{x}) = \max_{j=1}^{K} g_j(\boldsymbol{x}) \,. \tag{10.8}$$

Wenn wir uns erinnern, dass $|g_i(\boldsymbol{x})|/\|\boldsymbol{w}_i\|$ die Distanz zwischen dem Eingabepunkt zur Hyperebene darstellt und dann annehmen, dass alle $\boldsymbol{w}_i$ von ähnlicher Länge sind, dann weist dies den Punkt der Klasse (von allen $g_j(\boldsymbol{x}) > 0$) zu, von deren Hyperebene der Punkt am weitesten entfernt ist. Hierbei spricht man von einem *linearen Klassifikator* und geometrisch gesehen zerlegt er den Merkmalsraum in K konvexe Entscheidungsregionen $\mathcal{R}_i$ (siehe Abbildung 10.3).

LINEARER KLASSIFIKATOR

10.4 Paarweise Trennung

Wenn die Klassen nicht linear getrennt werden können, dann besteht eine Herangehensweise darin, das Ganze in eine Menge linearer Probleme zu zerlegen. Eine Möglichkeit ist die *paarweise Trennung* von Klassen (Duda, Hart und Stork 2001). Sie nutzt $K(K-1)/2$ lineare Diskriminanzfunktionen $g_{ij}(\boldsymbol{x})$, und zwar eine für jedes Paar von verschiedenen

PAARWEISE TRENNUNG

Klassen (siehe Abbildung 10.4):

$$g_{ij}(\boldsymbol{x}|\boldsymbol{w}_{ij}, w_{ij0}) = \boldsymbol{w}_{ij}^T\boldsymbol{x} + w_{ij0} \,.$$

Die Parameter $\boldsymbol{w}_{ij}, j \neq i$ werden während der Trainingsphase berechnet, so dass sich ergibt

$$g_{ij}(\boldsymbol{x}) = \begin{cases} > 0 \;\text{ falls } \boldsymbol{x} \in \mathcal{C}_i \,, \\ \leq 0 \;\text{ falls } \boldsymbol{x} \in \mathcal{C}_j \,, \; i, j = 1, \ldots, K \text{ und } i \neq j \,. \\ \text{egal andernfalls} \,, \end{cases} \tag{10.9}$$

Das heißt, falls $\boldsymbol{x}^t \in \mathcal{C}_k$, wobei $k \neq i, k \neq j$, dann wird $\boldsymbol{x}^t$ während der Übung von $g_{ij}(\boldsymbol{x})$ nicht verwendet.

Während der Testphase folgen wir der Anweisung:

$$\text{wähle } \mathcal{C}_i, \text{ falls } \forall j \neq i, g_{ij}(\boldsymbol{x}) > 0 \,.$$

In vielen Fällen ist dies möglicherweise nicht für jedes beliebige i zutreffend, und wenn wir solche Fälle nicht ablehnen wollen, dann können wir die Konjunktion abschwächen, indem wir eine Summation ausführen und das Maximum bilden:

$$g_i(\boldsymbol{x}) = \sum_{j \neq i} g_{ij}(\boldsymbol{x}) \,. \tag{10.10}$$

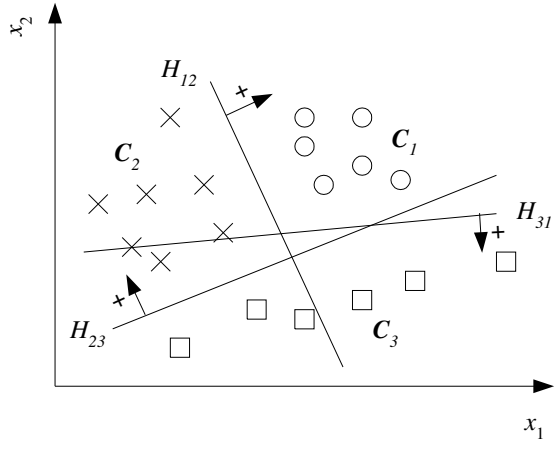


Abb. 10.4: Bei der paarweisen linearen Trennung gibt es eine eigene Hyperebene für jedes Paar von Klassen. Damit eine Eingabe $\mathcal{C}_1$ zugewiesen wird, sollte sie auf der positiven Seite von H_{12} und H_{13} (was die negative Seite von H_{31} ist) liegen. Der Wert von H_{23} kümmert uns nicht. In diesem Fall ist $\mathcal{C}_1$ zwar nicht linear trennbar von anderen Klassen, kann aber paarweise linear getrennt werden.

Wenn die Klassen zwar nicht linear trennbar aber paarweise linear trennbar sind – letzteres ist viel wahrscheinlicher – dann kann das Vorgehen der paarweisen Trennung genutzt werden, was zur nichtlinearen Trennung von Klassen führt (siehe Abbildung 10.4). Dies ist ein weiteres Beispiel für das Herunterbrechen eines komplexen, beispielsweise nichtlinearen Problems, in eine Menge einfacherer, beispielsweise linearer Probleme. Wir haben bereits Entscheidungsbäume betrachtet (Kapitel 9), welche diese Grundidee nutzen, und wir werden weitere Beispiele hierfür in Kapitel 17 diskutieren, wenn es um die Kombination multipler Modelle, zum Beispiel Fehlerkorrekturcodes, und gemischte Expertenmodelle geht, bei denen die Anzahl an linearen Modellen kleiner als $\mathcal{O}(K^2)$ ist.

10.5 Neubetrachtung der parametrischen Diskriminanz

In Kapitel 5 haben wir gesehen, dass bei gaußverteilten Klassendichten $p(\boldsymbol{x}|\mathcal{C}_i)$, die eine gemeinsame Kovarianzmatrix besitzen, die Diskriminanzfunktion linear ist:

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} \,, \tag{10.11}$$

wobei die Parameter analytisch berechnet werden können als

$$\begin{aligned} \boldsymbol{w}_i &= \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \,, \\ w_{i0} &= -\frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \log P(\mathcal{C}_i) \,. \end{aligned} \tag{10.12}$$

Bei einem gegebenen Datensatz berechnen wir zuerst die Schätzungen für $\boldsymbol{\mu}_i$ und $\boldsymbol{\Sigma}$, setzen dann die Schätzungen, $\boldsymbol{m}_i$, $\mathbf{S}$, in Gleichung 10.12 ein und berechnen die Parameter der linearen Diskriminanzfunktion.

Betrachten wir noch einmal den Spezialfall, dass zwei Klassen vorliegen. Wir definieren $y \equiv P(\mathcal{C}_1|\boldsymbol{x})$ und $p(\mathcal{C}_2|\boldsymbol{x}) = 1 - y$. Bei der Klassifikation gilt dann:

$$\text{wähle } \mathcal{C}_1, \text{ falls } \begin{cases} y > 0{,}5 \,, \\ \dfrac{y}{1-y} > 1 \,, \\ \log \dfrac{y}{1-y} > 0 \,, \end{cases} \quad \text{und andernfalls } \mathcal{C}_2 \,.$$

$\log y/(1-y)$ ist bekannt als die *Logit-Transformation* oder als *Log Odds* (dt. *logarithmierte Chancenverhältnisse*) von y. Im Falle von zwei nor- LOGIT-TRANSFORMATION

malverteilten Klassen, die eine gemeinsame Kovarianzmatrix besitzen, sind die Log Odds linear:

Log Odds

$$\begin{aligned}
\text{logit}(P(\mathcal{C}_1|\boldsymbol{x})) &= \log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{1-P(\mathcal{C}_1|\boldsymbol{x})} = \log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{P(\mathcal{C}_2|\boldsymbol{x})} \\
&= \log \frac{p(\boldsymbol{x}|\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)} + \log \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \\
&= \log \frac{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2} \exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)]}{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2} \exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)]} \\
&\quad + \log \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \\
&= \boldsymbol{w}^T \boldsymbol{x} + w_0 \,, \qquad (10.13)
\end{aligned}$$

mit

$$\begin{aligned}
\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \,, \\
w_0 &= -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \log \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \,. \qquad (10.14)
\end{aligned}$$

Die Inversion von Logit,

$$\log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{1-P(\mathcal{C}_1|\boldsymbol{x})} = \boldsymbol{w}^T \boldsymbol{x} + w_0 \,,$$

logistische Funktion

ist die *logistische Funktion*, auch bekannt als die *Sigmoidfunktion* (siehe Abbildung 10.5):

Sigmoidfunktion

$$P(\mathcal{C}_1|\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x} + w_0) = \frac{1}{1+\exp\left[-(\boldsymbol{w}^T \boldsymbol{x} + w_0)\right]} \,. \qquad (10.15)$$

Während der Trainingsphase schätzen wir $\boldsymbol{m}_1, \boldsymbol{m}_2, \mathbf{S}$ und setzen diese Werte in Gleichung 10.14 ein, um die Diskriminanzparameter zu berechnen. Während der Testphase und bei gegebenem $\boldsymbol{x}$ wählen wir einen der folgenden Schritte:

1. Berechne $g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$ und wähle $\mathcal{C}_1$, falls $g(\boldsymbol{x}) > 0$, oder
2. Berechne $y = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x} + w_0)$ und wähle $\mathcal{C}_1$, falls $y > 0{,}5$,

da sigmoid(0) = 0,5. In letzterem Fall formt die Sigmoidfunktion den Diskriminanzwert in eine a-posteriori-Wahrscheinlichkeit um. Dies ist zulässig, wenn es zwei Klassen und eine Diskriminanzfunktion gibt; wir werden in Abschnitt 10.7 lernen, wie wir die a-posteriori-Wahrscheinlichkeiten für $K > 2$ schätzen können.

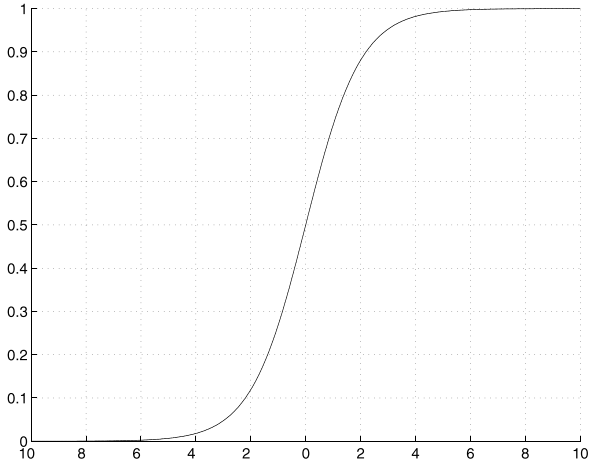


Abb. 10.5: Die logistische Funktion, auch bekannt als Sigmoidfunktion.

10.6 Gradientenabstieg

Bei der Likelihood-basierten Klassifikation waren die Parameter die ausreichenden Statistiken von $p(\boldsymbol{x}|\mathcal{C}_i)$ und $P(\mathcal{C}_i)$, und die Methode, die wir zur Schätzung der Parameter verwendeten, war die der Maximum-Likelihood. Im diskriminanzbasierten Ansatz nutzen wir die Parameter der Diskriminanzfunktionen und optimieren sie, um den Klassifikationsfehler am Datensatz zu minimieren. Wenn $\boldsymbol{w}$ die Parametermenge darstellt und $E(\boldsymbol{w}|\mathcal{X})$ der Fehler mit den Parametern $\boldsymbol{w}$ am gegebenen Satz $\mathcal{X}$ ist, so suchen wir nach

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} E(\boldsymbol{w}|\mathcal{X}) \,.$$

In vielen Fällen – von denen wir einige in Kürze betrachten werden – gibt es keine analytische Lösung und wir müssen auf iterative Optimierungsmethoden zurückgreifen, wobei die am häufigsten verwendete die des *Gradientenabstiegs* ist. Sei $E(\boldsymbol{w})$ eine differenzierbare Funktion eines Vektors von Variablen, dann besteht der *Gradientenvektor* aus den partiellen Ableitungen

Gradientenabstieg
Gradientenvektor

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d}\right]^T \,.$$

Die Prozedur des *Gradientenabstiegs* für die Minimierung von E startet mit einem zufälligen $\boldsymbol{w}$, und aktualisiert $\boldsymbol{w}$ bei jedem Schritt in

entgegengesetzter Richtung zum Gradienten

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i \,, \tag{10.16}$$

$$w_i = w_i + \Delta w_i \,, \tag{10.17}$$

wobei η als *Schrittweite* oder *Lernfaktor* bezeichnet wird und bestimmt, wie weit in die jeweilige Richtung zu gehen ist. Der Gradientenanstieg wird genutzt, um eine Funktion zu maximieren und bewegt sich in Richtung des Gradienten. Wenn wir ein Minimum (oder Maximum) erreichen, ist die Ableitung gleich 0 und die Prozedur wird beendet. Dies deutet darauf hin, dass die Prozedur das naheliegendste Minimum findet, welches allerdings ein lokales Minimum sein könnte; es gibt keinerlei Garantie dafür, dass auch das globale Minimum gefunden werden kann, es sei denn, die Funktion besitzt nur ein Minimum. Die Nutzung eines entsprechend gut gewählten Wertes für η ist ebenfalls von Bedeutung: ist er zu klein, kommt es nicht schnell genug zur Konvergenz; ein großer Wert könnte Oszillationen und sogar Divergenz verursachen.

In diesem Buch nutzen wir Gradientenmethoden, die einfach und recht effektiv sind. Wir bedenken allerdings, dass nach vollendeter Definition eines geeigneten Modells und einer Fehlerfunktion die Optimierung der Modellparameter zur Minimierung der Fehlerfunktion vollzogen werden kann, indem eine von zahlreichen möglichen Techniken gewählt wird. Es existieren Methoden zweiter Ordnung und konjugierter Gradienten, die schneller konvergieren, jedoch auf Kosten von mehr Speicherplatz und Rechenleistung. Kostspieligere Methoden wie das Simulated Annealing und genetische Algorithmen erlauben eine umfassendere Suche im Parameterraum und sind weniger abhängig vom Ausgangspunkt.

10.7 Logistische Diskriminanz

10.7.1 Zwei Klassen

LOGISTISCHE DISKRIMINANZ

Bei der *logistischen Diskriminanz* modellieren wir nicht die klassenbezogenen Verteilungen $p(\boldsymbol{x}|\mathcal{C}_i)$, sondern stattdessen deren Verhältnis. Beginnen wir erneut mit zwei Klassen und nehmen wir an, dass das Log-Likelihood-Verhältnis linear ist:

$$\log \frac{p(\boldsymbol{x}|\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)} = \boldsymbol{w}^T \boldsymbol{x} + w_0^o \,. \tag{10.18}$$

Dies trifft tatsächlich zu, wenn die klassenbezogenen Dichten normalverteilt sind (Gleichung 10.13). Die logistische Diskriminanz hat jedoch einen vielfältigeren Anwendungsbereich; beispielsweise könnte $\boldsymbol{x}$ aus diskreten

Attributen bestehen oder als Mischung kontinuierlicher und diskreter Attribute vorliegen.

Unter Zuhilfenahme des Satzes von Bayes erhalten wir

$$\begin{aligned}\text{logit}(P(\mathcal{C}_1|\boldsymbol{x})) &= \log \frac{P(\mathcal{C}_1|\boldsymbol{x})}{1 - P(\mathcal{C}_1|\boldsymbol{x})} \\ &= \log \frac{p(\boldsymbol{x}|\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)} + \log \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \\ &= \boldsymbol{w}^T \boldsymbol{x} + w_0 \,, \end{aligned} \tag{10.19}$$

wobei

$$w_0 = w_0^o + \log \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \,. \tag{10.20}$$

Formen wir die Terme um, erhalten wir wieder die sigmoidale Funktion

$$y = \hat{P}(\mathcal{C}_1|\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T \boldsymbol{x} + w_0)]} \tag{10.21}$$

als unseren Schätzer für $P(\mathcal{C}_1|\boldsymbol{x})$.

Sehen wir uns nun einmal an, wie wir $\boldsymbol{w}$ und w_0 lernen können. Uns liegt eine Stichprobe von zwei Klassen vor, $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ mit $r^t = 1$, falls $\boldsymbol{x} \in \mathcal{C}_1$, und $r^t = 0$, falls $\boldsymbol{x} \in \mathcal{C}_2$. Wir nehmen an, dass r^t bei gegebenem $\boldsymbol{x}^t$ eine Bernoulli-Verteilung mit der Wahrscheinlichkeit $y^t \equiv P(\mathcal{C}_1|\boldsymbol{x}^t)$ ist, wie in Gleichung 10.21 berechnet:

$$r^t|\boldsymbol{x}^t \sim \text{Bernoulli}(y^t) \,.$$

Hier wird der Unterschied zu den Likelihood-basierten Methoden deutlich, bei denen wir $p(\boldsymbol{x}|\mathcal{C}_i)$ modelliert haben; beim diskriminanzbasierten Ansatz modellieren wir $r|\boldsymbol{x}$ direkt. Die Likelihood der Stichprobe ist

$$l(\boldsymbol{w}, w_0|\mathcal{X}) = \prod_t (y^t)^{(r^t)} (1 - y^t)^{(1-r^t)} \,. \tag{10.22}$$

Wie wir wissen, können wir eine zu maximierende Likelihoodfunktion immer in eine zu minimierende Fehlerfunktion umformen, $E = -\log l$, und in unserem Fall liegt *Kreuzentropie* vor: KREUZENTROPIE

$$E(\boldsymbol{w}, w_0|\mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) \,. \tag{10.23}$$

Wegen der Nichtlinearität der Sigmoidfunktion ist eine direkte Lösung nicht möglich. Wir nutzen den Gradientenabstieg, um die Kreuzentropie

zu minimieren, äquivalent zur Maximierung der Likelihood oder Log-Likelihood. Falls $y = \text{sigmoid}(a) = 1/(1 + \exp(-a))$, ist die Ableitung gegeben als

$$\frac{dy}{da} = y(1 - y)$$

und wir erhalten die folgenden Aktualisierungsgleichungen:

$$\begin{aligned}
\Delta w_j &= -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left(\frac{r^t}{y^t} - \frac{1 - r^t}{1 - y^t} \right) y^t (1 - y^t) x_j^t \\
&= \eta \sum_t (r^t - y^t) x_j^t, j = 1, \ldots, d\,, \\
\Delta w_0 &= -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^t - y^t)\,. \qquad (10.24)
\end{aligned}$$

Es ist ratsam, w_j mit zufälligen Werten nahe 0 zu initialisieren; generell werden diese einheitlich aus dem Intervall $[-0{,}01, 0{,}01]$ gezogen. Der Grund dafür ist der, dass bei sehr großen initialen w_j die gewichtete Summe ebenfalls sehr groß ausfallen und damit die Sigmoidfunktion sättigen kann. Wir erkennen anhand von Abbildung 10.5, dass bei Anfangsgewichtungen nahe 0 die Summe innerhalb der Mittelregion bleibt, wo die Ableitung ungleich 0 ist und eine Aktualisierung durchgeführt werden kann. Liegt die gewichtete Summe in hohen Größenordnungen vor (kleiner -5 oder größer als $+5$), so ist die Ableitung der Sigmoidfunktion fast 0 und Gewichte werden nicht aktualisiert.

Der Pseudocode ist im Listing 10.1 zu sehen. Ein Beispiel mit einer eindimensionalen Eingabe findet sich in Abbildung 10.6. Sowohl die Gerade $wx + w_0$ als auch ihr Wert nach der Sigmoidfunktion sind als Funktion von Lerniterationen abgebildet. Wir erkennen, dass sich die Sigmoidfunktion stärker herausbildet, um Ausgaben von 0 und 1 zu erhalten, was durch Erhöhung des Betrags von w bzw. im multivariaten Fall von $\|\boldsymbol{w}\|$ zu erreichen ist.

Sobald das Training vollendet ist und wir die finalen $\boldsymbol{w}$ und w_0 haben, berechnen wir bei gegebenem $\boldsymbol{x}^t$ während der Testphase $y^t = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x}^t + w_0)$ und wir wählen $\mathcal{C}_1$, falls $y^t > 0{,}5$ bzw. in den anderen Fällen $\mathcal{C}_2$. Dies impliziert, dass wir, um die Zahl an Fehlklassifikationen zu minimieren, den Lernprozess nicht solange fortführen müssen, bis y^t gleich 0 oder 1 ist, sondern nur bis y^t kleiner oder größer als 0,5 ist und somit auf der richtigen Seite der Entscheidungsgrenze liegt. Wenn wir das Training jenseits dieses Zeitpunktes fortführen, wird die Kreuzentropie weiter abnehmen ($|w_j|$ wird weiter steigen, um die Sigmoidfunktion zu festigen), doch die Zahl an Fehlklassifikationen wird sich nicht mehr verringern. Generell setzen wir das Training fort, bis die Zahl an Fehlklassifikationen sich nicht mehr verringert (der Wert

wird 0 sein, falls die Klassen linear separierbar sind). Tatsächlich ist das *frühe Stoppen* vor dem Erreichen eines Trainingsfehlers von 0 eine Form der Regularisierung. Da wir mit Gewichten nahe 0 starten und diese sich dann mit fortschreitendem Training von 0 entfernen, führt frühes Stoppen zu einem Modell mit vielen Gewichten nahe 0 und effektiv weniger Parametern.

FRÜHES STOPPEN

Folgender Hinweis sei noch gegeben: um die Diskriminanzfunktion abzuleiten, nahmen wir zwar an, dass die Log-Verhältnisse der Klassendichten linear sind, jedoch schätzen wir die a-posteriori-Verteilungen direkt, wohingegen wir $p(\boldsymbol{x}|\mathcal{C}_i)$ oder $P(\mathcal{C}_i)$ niemals explizit schätzen.

10.7.2 Multiple Klassen

Verallgemeinern wir nun auf $K > 2$ Klassen. Wir wählen eine der Klassen, beispielsweise $\mathcal{C}_K$, als Referenzklasse und nehmen an, dass

$$\log \frac{p(\boldsymbol{x}|\mathcal{C}_i)}{p(\boldsymbol{x}|\mathcal{C}_K)} = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}^o \,. \tag{10.25}$$

Dann erhalten wir

$$\frac{P(\mathcal{C}_i|\boldsymbol{x})}{P(\mathcal{C}_K|\boldsymbol{x})} = \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}] \tag{10.26}$$

```
For j = 0, ..., d
    w_j ←rand(-0,01, 0,01)
Repeat
    For j = 0, ..., d
        Δw_j ← 0
    For t = 1, ..., N
        o ← 0
        For j = 0, ..., d
            o ← o + w_j x_j^t
        y ← sigmoid(o)
    For j = 0, ..., d
        Δw_j ← Δw_j + (r^t − y) x_j^t
    For j = 0, ..., d
        w_j ← w_j + η Δw_j
Until Konvergenz
```

***Listing 10.1*:** Algorithmus zur logistischen Diskriminanz, der den Gradientenabstieg für den Fall einer einzelnen Ausgabe bei zwei Klassen implementiert. Für w_0 nehmen wir an, dass es eine zusätzliche Eingabe x_0 gibt, für die immer gilt $+1$: $x_0^t \equiv +1, \forall t$.

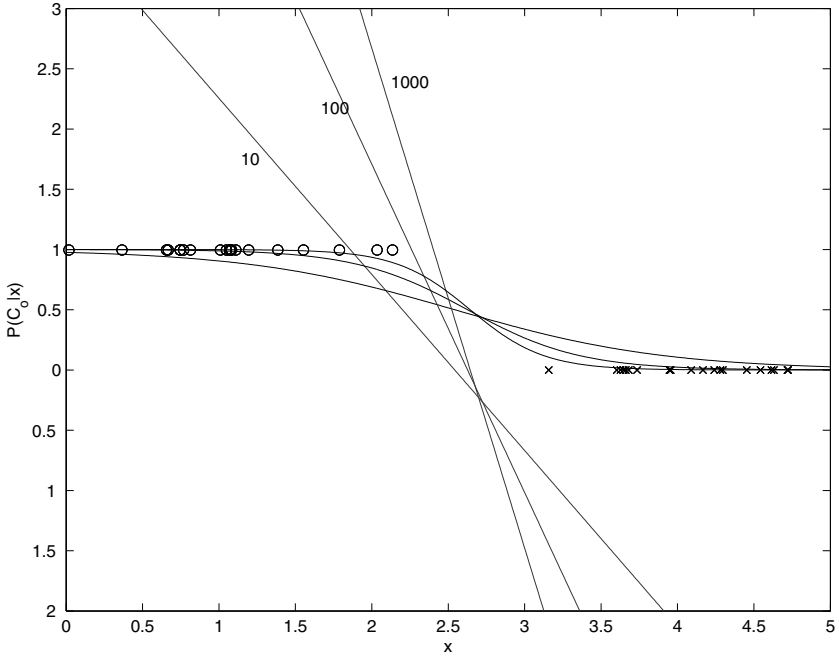


***Abb. 10.6*:** Für ein univariates Zweiklassenproblem (repräsentiert durch „○“ und „×“) sind die Entwicklungen der Geraden $wx + w_0$ und die sigmoidale Ausgabe nach 10, 100 und 1.000 Iterationen über die Stichprobe hinweg dargestellt.

mit $w_{i0} = w_{i0}^o + \log P(\mathcal{C}_i)/P(\mathcal{C}_K)$. Wir erkennen, dass

$$\sum_{i=1}^{K-1} \frac{P(\mathcal{C}_i|\boldsymbol{x})}{P(\mathcal{C}_K|\boldsymbol{x})} = \frac{1 - P(\mathcal{C}_K|\boldsymbol{x})}{P(\mathcal{C}_K|\boldsymbol{x})} = \sum_{i=1}^{K-1} \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]$$

$$\Rightarrow P(\mathcal{C}_K|\boldsymbol{x}) = \frac{1}{1 + \sum_{i=1}^{K-1} \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]} \tag{10.27}$$

sowie

$$\frac{P(\mathcal{C}_i|\boldsymbol{x})}{P(\mathcal{C}_K|\boldsymbol{x})} = \exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]$$

$$\Rightarrow P(\mathcal{C}_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]}{1 + \sum_{j=1}^{K-1} \exp[\boldsymbol{w}_j^T \boldsymbol{x} + w_{j0}]},\ i = 1, \ldots, K-1\,. \tag{10.28}$$

Um alle Klassen einheitlich zu behandeln, können wir schreiben:

$$y_i = \hat{P}(\mathcal{C}_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}]}{\sum_{j=1}^{K} \exp[\boldsymbol{w}_j^T \boldsymbol{x} + w_{j0}]},\ i = 1, \ldots, K\,, \tag{10.29}$$

was man als *Softmax-Funktion* bezeichnet (Bridle 1990). Ist die gewichtete Summe für eine Klasse signifikant größer als für die anderen Klassen, und zwar nachdem sie exponiert und normalisiert wurde, so liegt das entsprechende y_i nahe 1 und die anderen Werte sind nahe 0. Somit funktioniert dies ähnlich der Berechnung eines Maximums, mit dem Unterschied, dass wir hier differenzieren können; daher auch der Name Softmax. Softmax garantiert ebenfalls, dass $\sum_i y_i = 1$. SOFTMAX

Schauen wir uns nun an, wie wir die Parameter lernen können. Im Falle von $K > 2$ Klassen entspricht jeder Punkt der Stichprobe einem multinomialen Versuch mit einer Ziehung, das heißt $\boldsymbol{r}^t|\boldsymbol{x}^t \sim \text{Mult}_k(1, \boldsymbol{y}^t)$, wobei $y_i^t \equiv P(\mathcal{C}_i|\boldsymbol{x}^t)$. Die Likelihood der Stichprobe ist

$$l(\{\boldsymbol{w}_i, w_{i0}\}_i|\mathcal{X}) = \prod_t \prod_i (y_i^t)^{r_i^t} \tag{10.30}$$

und die Fehlerfunktion wird erneut durch die Kreuzentropie gegeben:

$$E(\{\boldsymbol{w}_i, w_{i0}\}_i|\mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t \,. \tag{10.31}$$

Wir benutzen wieder das Gradientenabstiegsverfahren und erhalten für $y_i = \exp(a_i)/\sum_j \exp(a_j)$

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_j) \,, \tag{10.32}$$

wobei δ_{ij} dem Kronecker-Delta entspricht, welches 1 ist, falls $i = j$ und 0, falls $i \neq j$ (Übung 3). Vorausgesetzt, dass $\sum_i r_i^t = 1$, erhalten wir die folgenden Aktualisierungsgleichungen für $j = 1, \ldots, K$

$$\begin{aligned}
\Delta \boldsymbol{w}_j &= \eta \sum_t \sum_i \frac{r_i^t}{y_i^t} y_i^t (\delta_{ij} - y_j^t) \boldsymbol{x}^t \\
&= \eta \sum_t \sum_i r_i^t (\delta_{ij} - y_j^t) \boldsymbol{x}^t \\
&= \eta \sum_t \left[\sum_i r_i^t \delta_{ij} - y_j^t \sum_i r_i^t \right] \boldsymbol{x}^t \\
&= \eta \sum_t (r_j^t - y_j^t) \boldsymbol{x}^t \,, \\
\Delta w_{j0} &= \eta \sum_t (r_j^t - y_j^t) \,.
\end{aligned} \tag{10.33}$$

Man beachte, dass aufgrund der Normalisierung in Softmax $\boldsymbol{w}_j$ und w_{j0} nicht nur durch $\boldsymbol{x}^t \in \mathcal{C}_j$ beeinflusst werden, sondern auch durch $\boldsymbol{x}^t \in \mathcal{C}_i, i \neq j$. Die Diskriminanzen werden aktualisiert, so dass die

```
For i = 1, ..., K
    For j = 0, ..., d
        w_ij ← rand(−0,01, 0,01)
Repeat
    For i = 1, ..., K, For j = 0, ..., d, Δw_ij ← 0
    For t = 1, ..., N
        For i = 1, ..., K
            o_i ← 0
            For j = 0, ..., d
                o_i ← o_i + w_ij x_j^t
        For i = 1, ..., K
            y_i ← exp(o_i)/ Σ_k exp(o_k)
        For i = 1, ..., K
            For j = 0, ..., d
                Δw_ij ← Δw_ij + (r_i^t − y_i) x_j^t
    For i = 1, ..., K
        For j = 0, ..., d
            w_ij ← w_ij + η Δw_ij
Until Konvergenz
```

***Listing 10.2*:** Die Abbildung zeigt den Algorithmus der logistischen Diskriminanz zur Implementierung des Gradientenabstiegs für den Fall mit $K > 2$ Klassen. Zur Verallgemeinerung nehmen wir $x_0^t \equiv 1, \forall t$.

korrekte Klasse die höchste gewichtete Summe nach der Anwendung von Softmax hat und die gewichteten Summen aller anderen Klassen so niedrige Werte wie möglich aufweisen. Der Pseudocode ist im Listing 10.2 gegeben. Für ein zweidimensionales Beispiel mit drei Klassen ist das Konturdiagramm in Abbildung 10.7 dargestellt und die Diskriminanzen und a-posteriori-Wahrscheinlichkeiten finden sich in Abbildung 10.8.

Während der Testphase berechnen wir alle $y_k, k = 1, \ldots, K$ und wählen $\mathcal{C}_i$, falls $y_i = \max_k y_k$. Wie wir bereits für $K = 2$ gesehen haben, muss das Training nicht fortgesetzt werden bis die Kreuzentropie so weit wie möglich minimiert ist. Es reicht das Training solange fortzusetzen, bis für jede Instanz der Trainingsmenge die korrekte Klasse die höchste gewichtete Summe besitzt. Wenn man also die Fehlklassifikationen nach jedem Schritt überprüft, kann das Training in diesem Fall früher beendet werden.

Wenn Daten normalverteilt sind, hat die logistische Diskriminanzfunktion eine Fehlerrate vergleichbar mit der parametrischen, auf Normalverteilung basierenden, linearen Diskriminanzfunktion (McLachlan 1992). Die logistische Diskriminanzfunktion kann auch dann noch genutzt werden, wenn die klassenbezogenen Dichten nicht normalverteilt oder wenn sie nicht unimodal sind, vorausgesetzt, die Klassen sind linear trennbar.

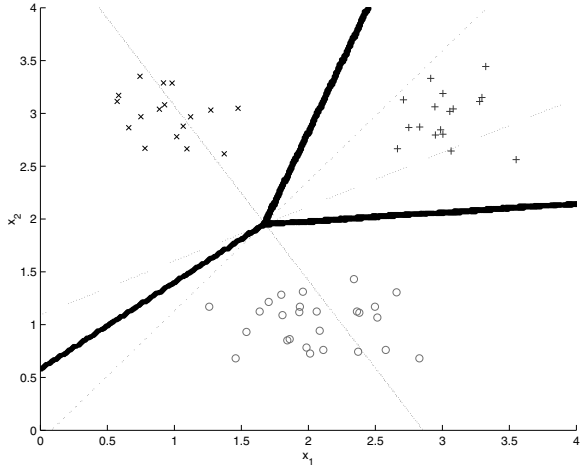


***Abb. 10.7*:** Dargestellt ist die durch die logistische Diskriminanzfunktion gefundene Lösung für ein zweidimensionales Problem mit drei Klassen. Die dünnen Linien geben an, wo $g_i(\boldsymbol{x}) = 0$, und die dicke Gerade entspricht der Grenze, welche durch den linearen Klassifikator, der das Maximum wählt, induziert wird.

Das Verhältnis der klassenbezogenen Dichten muss natürlich nicht zwangsläufig linear sein (Anderson 1982; McLachlan 1992). Unter der Annahme einer quadratischen Diskriminanzfunktion erhalten wir

$$\log \frac{p(\boldsymbol{x}|\mathcal{C}_i)}{p(\boldsymbol{x}|\mathcal{C}_k)} = \boldsymbol{x}^T \mathbf{W}_i \boldsymbol{x} + \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0} \,, \tag{10.34}$$

was der parametrischen Diskriminanz mit multivariaten, normalverteilten klassenbezogenen Dichten mit unterschiedlichen Kovarianzmatrizen entspricht und dies generalisiert. Genauso wie wir $\boldsymbol{\Sigma}_i$ vereinfachen (regulieren) können, ist es bei großem d auch möglich, das gleiche mit $\mathbf{W}_i$ zu tun, indem wir nur seine führenden Eigenvektoren in die Betrachtung einbeziehen.

Wie in Abschnitt 10.2 besprochen, kann jede beliebige Funktion der Basisvariablen als zusätzliche Variation dieser als weitere Eingabevariable verwendet werden. Zum Beispiel kann man die Diskriminanzfunktion als lineare Summe von nichtlinearen Basisfunktionen schreiben,

$$\log \frac{p(\boldsymbol{x}|\mathcal{C}_i)}{p(\boldsymbol{x}|\mathcal{C}_k)} = \boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}) + w_{i0} \,, \tag{10.35}$$

wobei $\boldsymbol{\phi}(\cdot)$ die Basisfunktionen repräsentiert, welche als transformierte Variablen angesehen werden können. In der Terminologie von neuronalen Netzen spricht man hierbei von *mehrlagigen Perzeptronen* (Kapitel 11) und die Sigmoidfunktion ist hier die populärste Basisfunktion. Wird

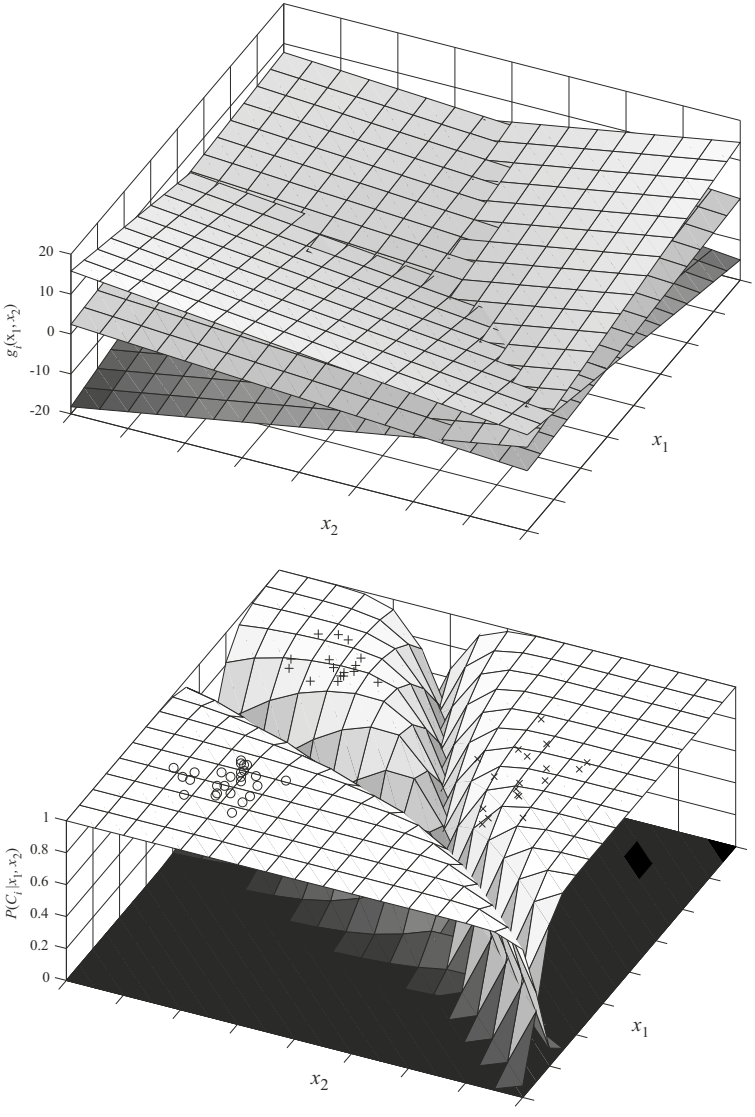


***Abb. 10.8*:** Für dasselbe Beispiel wie in Abbildung 10.7 sind die linearen Diskriminanzen (oben) und die a-posteriori-Wahrscheinlichkeiten nach Anwendung von Softmax (unten) dargestellt.

eine Gaußsche Basisfunktion verwendet, bezeichnet man das Modell als *radiale Basisfunktionen* (Kapitel 12). Wir können sogar einen komplett nichtparametrischen Ansatz verwenden, zum Beispiel das Parzen-Fenster (Kapitel 8).

10.8 Diskriminanz durch Regression

Bei der Regression ist das probabilistische Modell

$$r^t = y^t + \epsilon \tag{10.36}$$

mit $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Falls $r^t \in \{0, 1\}$, kann y^t mit Hilfe der Sigmoidfunktion auf nur diesen Bereich eingeschränkt werden. Unter der Annahme eines linearen Modells und zweier Klassen erhalten wir

$$y^t = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x}^t + w_0) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T \boldsymbol{x}^t + w_0)]} . \tag{10.37}$$

Gehen wir dann davon aus, dass $r|\boldsymbol{x} \sim \mathcal{N}(y, \sigma^2)$, so ist die Likelihood der Stichprobe bei der Regression gleich

$$l(\boldsymbol{w}, w_0|\mathcal{X}) = \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(r^t - y^t)^2}{2\sigma^2}\right] . \tag{10.38}$$

Die Maximierung der Log-Likelihood entspricht der Minimierung der Summe der quadrierten Fehler

$$E(\boldsymbol{w}, w_0|\mathcal{X}) = \frac{1}{2} \sum_t (r^t - y^t)^2 . \tag{10.39}$$

Nutzen wir den Gradientenabstieg, ergibt sich

$$\begin{aligned} \Delta \boldsymbol{w} &= \eta \sum_t (r^t - y^t) y^t (1 - y^t) \boldsymbol{x}^t , \\ \Delta w_0 &= \eta \sum_t (r^t - y^t) y^t (1 - y^t) . \end{aligned} \tag{10.40}$$

Diese Methode kann auch verwendet werden, wenn es $K > 2$ Klassen gibt. Das probabilistische Modell ist

$$\boldsymbol{r}^t = \boldsymbol{y}^t + \boldsymbol{\epsilon} \tag{10.41}$$

mit $\boldsymbol{\epsilon} \sim \mathcal{N}_K(0, \sigma^2 \mathbf{I}_K)$. Unter der Annahme eines linearen Modells für jede Klasse erhalten wir

$$y_i^t = \text{sigmoid}(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_i^T \boldsymbol{x}^t + w_{i0})]} . \tag{10.42}$$

Dann ist die Likelihood der Stichprobe

$$l(\{\boldsymbol{w}_i, w_{i0}\}_i|\mathcal{X}) = \prod_t \frac{1}{(2\pi)^{K/2}|\Sigma|^{1/2}} \exp\left[-\frac{\|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2}{2\sigma^2}\right] \quad (10.43)$$

und die Fehlerfunktion ist

$$\begin{aligned} E(\{\boldsymbol{w}_i, w_{i0}\}_i|\mathcal{X}) &= \frac{1}{2}\sum_t \|\boldsymbol{r}^t - \boldsymbol{y}^t\|^2 \\ &= \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \,. \end{aligned} \quad (10.44)$$

Die Aktualisierungsgleichungen für $i = 1, \ldots, K$ sind

$$\begin{aligned} \Delta\boldsymbol{w}_i &= \eta \sum_t (r_i^t - y_i^t) y_i^t (1 - y_i^t) \boldsymbol{x}^t \,, \\ \Delta w_{i0} &= \eta \sum_t (r_i^t - y_i^t) y_i^t (1 - y_i^t) \,. \end{aligned} \quad (10.45)$$

Man beachte jedoch, dass wir auf diese Weise die Information außen vor lassen, dass nur eins der y_i 1 sein muss und alle anderen 0 sind bzw. dass $\sum_i y_i = 1$. Die Softmax-Funktion aus Gleichung 10.29 erlaubt uns, diese zusätzliche Information einzuarbeiten, da die Ausgaben die a-posteriori-Wahrscheinlichkeiten der Klassen schätzen. Unter Nutzung von Ausgaben der Sigmoidfunktion im Fall von $K > 2$ behandeln wir die y_i, als ob sie unabhängige Funktionen wären.

Auf Folgendes sei ebenfalls noch hingewiesen: wenn wir den Regressionsansatz wählen, dann gibt es für eine vorliegende Klasse so lange Aktualisierungen, bis die richtige Ausgabe gleich 1 ist und alle anderen gleich 0 sind. Tatsächlich ist dies nicht notwendig, denn während der Testphase wählen wir ohnehin das Maximum; es reicht aus, die Trainingsphase so lange fortzuführen, bis die richtige Ausgabe größer ist als alle anderen, was genau dem entspricht, was die Softmax-Funktion bewirkt.

Somit ist dieser Ansatz mit multiplen Ausgaben der Sigmoidfunktion angemessener, wenn die Klassen sich *nicht* gegenseitig ausschließen und nicht vollständig sind. Das heißt, für ein $\boldsymbol{x}^t$ können alle r_i^t gleich 0 sein, sprich $\boldsymbol{x}^t$ gehört zu keiner der Klassen, oder mehr als ein r_i^t kann gleich 1 sein, falls die Klassen sich überlappen.

10.9 Lernen von Rangordnungen

RANKING Das Erstellen einer *Rangordnung* (engl. *Ranking*) ist ein Anwendungsgebiet des maschinellen Lernens, das sich von der Klassifikation wie

auch der Regression unterscheidet und in gewisser Weise zwischen diesen beiden Problemstellungen steht. Während es bei der Klassifikation und der Regression eine Eingabe $\boldsymbol{x}^t$ und eine gewünschte Ausgabe r^t gibt, geht es beim Ranking darum, zwei oder mehr Instanzen in die richtige Reihenfolge zu bringen (Liu 2011).

Nehmen wir zum Beispiel an, dass $\boldsymbol{x}^u$ und $\boldsymbol{x}^v$ zwei Filme repräsentieren und dass einem bestimmten Nutzer u besser gefallen hat als v (in diesem Fall müssen wir Filmen, die ähnlich sind wie u ein höheres Ranking geben). Dies wird durch $r^u \prec r^v$ gekennzeichnet. Was wir lernen, ist keine Diskriminanz- oder Regressionsfunktion, sondern eine Rangfunktion oder *Score-Funktion* $g(\boldsymbol{x}|\theta)$. Wichtig sind in diesem Fall nicht die Absolutwerte von $g(\boldsymbol{x}^u|\theta)$ und $g(\boldsymbol{x}^v|\theta)$, sondern wir müssen $\boldsymbol{x}^u$ einen höheren Score geben als $\boldsymbol{x}^v$, d. h., es muss $g(\boldsymbol{x}^u|\theta) > g(\boldsymbol{x}^v|\theta)$ für alle solchen Paare u und v gelten.

Wie üblich gehen wir von einem bestimmten Modell $g(\cdot)$ aus und optimieren dessen Parameter so, dass alle Rangbedingungen erfüllt sind. Um dann zum Beispiel einen Film zu empfehlen, den der Nutzer noch nicht gesehen hat, wählen wir den Film mit dem höchsten Score:

$$\text{wähle } u \text{ falls } g(\boldsymbol{x}^u|\theta) = \max_t g(\boldsymbol{x}^t|\theta) \,.$$

Manchmal wollen wir nicht nur die Instanz mit dem höchsten Score, sondern eine Liste der k höchsten.

An dieser Stelle sei auf die Vorteile und Eigenheiten von Rangordnern hingewiesen. Wenn die Nutzer die Filme, die sie gesehen haben, mit „hat mir gefallen“ und „hat mir nicht gefallen“ bewerten, dann entspricht dies einem Zweiklassenproblem und es kann ein Klassifikator benutzt werden. Aber Geschmack hat viele Nuancen, weshalb ein binäres Ranking unbefriedigend ist. Wenn die Nutzer dagegen den Grad ihres Gefallens (oder Nichtgefallens) auf einer Skala ausdrücken können, beispielsweise von eins bis zehn, entspricht dies einem Regressionsproblem. In diesem Fall entsteht die Ungenauigkeit dadurch, dass es schwierig ist, solche numerischen Werte objektiv zuzuordnen. Für Menschen ist es viel natürlicher zu beurteilen, welchen von zwei Filmen, die sie gesehen haben, sie besser fanden, anstatt für jeden eine ja/nein-Entscheidung zu treffen oder einen numerischen Wert für den Grad der Zustimmung anzugeben.

Das Rangordnungsproblem hat viele Anwendungen. Von Suchmaschinen zum Beispiel erwarten wir, dass sie diejenigen Dokumente finden, die für unsere Suchanfrage am relevantesten sind. Wenn die aktuell relevantesten Suchergebnisse angezeigt werden, der Nutzer aber die ersten beiden überspringt und stattdessen das dritte anklickt, ist das ein Hinweis darauf, dass das dritte Ergebnis ein höheres Ranking haben sollte als die aktuell ersten beiden. Solche Klickaufzeichnungen werden verwendet, um Ranking-Algorithmen zu trainieren.

Manchmal wird eine Neuordnung der Rangliste vorgenommen, um die Ausgabe des Ranking-Algorithmus durch neue Informationen zu verbessern. Bei der Spracherkennung zum Beispiel kann zunächst ein akustisches Modell verwendet werden, um eine geordnete Liste möglicher Sätze zu generieren. Anschließend können dann die N besten Kandidaten mithilfe von Merkmalen aus einem Sprachmodell in eine neue Rangliste gebracht werden, wodurch sich die Genauigkeit signifikant verbessern lässt (Shen und Joshi 2005).

Es gibt eine Reihe von Möglichkeiten, um einen Ranking-Algorithmus zu trainieren. Für alle Paare (u, v), für die $r^u \prec r^v$ definiert ist, haben wir einen Fehler, wenn $g(\boldsymbol{x}^v|\theta) > g(\boldsymbol{x}^u|\theta)$. Im Allgemeinen haben wir keine vollständiges Ranking für alle N^2 Paare, sondern nur für eine Teilmenge, wodurch eine Partialordnung definiert ist. Die Summe der Differenzen bildet den Fehler:

$$E(\boldsymbol{w}|\{r^u, r^v\}) = \sum_{r^u \prec r^v} \left[g(\boldsymbol{x}^v|\theta) - g(\boldsymbol{x}^u|\theta)\right]_+ . \tag{10.46}$$

Dabei ist a_+ gleich a, falls $a \geq 0$, andernfalls 0.

Wir wollen annehmen, dass wir, wie auch sonst in diesem Kapitel, ein lineares Modell verwenden:

$$g(\boldsymbol{x}|\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} . \tag{10.47}$$

Da wir uns nicht um die Absolutwerte kümmern, benötigen wir w_0 nicht. Der durch Gleichung 10.46 gegebene Fehler wird dann zu

$$E(\boldsymbol{w}|\{r^u, r^v\}) = \sum_{r^u \prec r^v} \left[\boldsymbol{w}^T(\boldsymbol{x}^v - \boldsymbol{x}^u)\right]_+ . \tag{10.48}$$

Mittels Gradientenabstieg können wir eine Online-Aktualisierung von $\boldsymbol{w}$ ausführen. Für jedes $r^u \prec r^v$, für das $g(\boldsymbol{x}^v|\theta) > g(\boldsymbol{x}^u|\theta)$ gilt, berechnen wir eine kleine Aktualisierung:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = -\eta \left(\boldsymbol{x}_j^v - \boldsymbol{x}_j^u\right), \quad j = 1, \ldots, d . \tag{10.49}$$

$\boldsymbol{w}$ ist so gewählt, dass sich bei Projektionen von Instanzen auf $\boldsymbol{w}$ das korrekte Ranking ergibt. Abbildung 10.9 zeigt Beispieldaten und die gelernten Projektionsrichtungen. Wie wir sehen, kann eine kleine Änderung der Ränge eine große Änderung von $\boldsymbol{w}$ bewirken.

Zur Verwendung von Fehlerfunktionen und Gradientenmethoden bei Rangordnungsproblemen einschließlich praktischer Anwendungen siehe Burges et al. 2005 sowie Shin und Josh 2005. Manchmal fordern wir für eine sichere Entscheidung für $r^u \prec r^v$, dass die Ausgabe nicht nur größer, sondern größer plus ein Margin ist, beispielsweise $g(\boldsymbol{x}^v|\theta) > 1 + g(\boldsymbol{x}^u|\theta)$. Wir werden ein Beispiel hierfür kennenlernen, wenn wir uns in Abschnitt 13.11 mit dem Lernen von Rangordnungen unter Verwendung von Kernel-Maschinen beschäftigen.

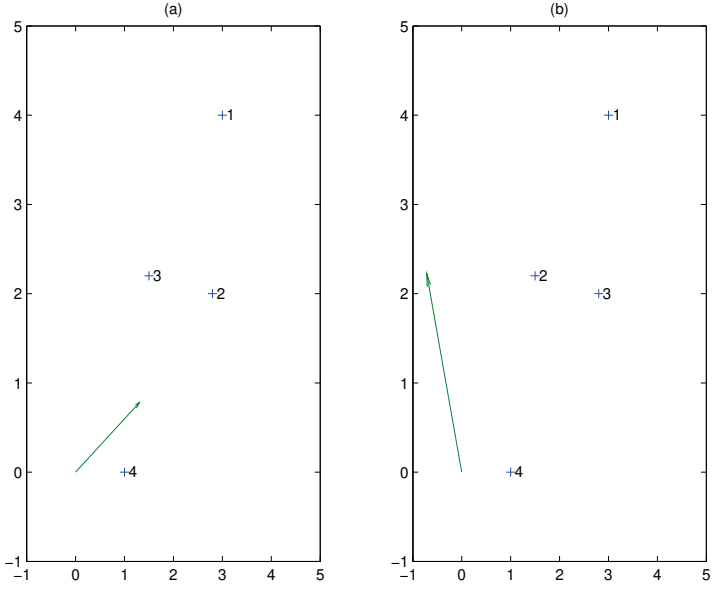


***Abb. 10.9*:** Rangordnungsprobleme und Lösungen. Die Datenpunkte sind durch ein + gekennzeichnet, und die danebenstehenden Zahlen geben den jeweiligen Rang an (wobei 1 der höchste Rang ist). Wir haben hier ein vollständiges Ranking. Der Pfeil zeigt das gelernte $\boldsymbol{w}$ an. Die Teile (a) und (b) zeigen zwei verschiedene Rangordnungsprobleme mit den zugehörigen Lösungen.

10.10 Anmerkungen

Die lineare Diskriminanzfunktion ist aufgrund ihrer Einfachheit der am häufigsten verwendete Klassifikator im Bereich der Mustererkennung (Duda, Hart und Stork 2001; McLachlan 1992). Wir haben den Fall der Gauß-Verteilung mit einer gemeinsamen Kovarianzmatrix in Kapitel 4 und Fishers lineare Diskriminanzfunktion in Kapitel 6 diskutiert; in diesem Kapitel nun untersuchten wir die logistische Diskriminanzfunktion. In Kapitel 11 werden wir Perzeptronen besprechen, was der Implementation einer linearen Diskriminanzfunktion als neuronales Netz entspricht. In Kapitel 13 beschäftigen wir uns mit Support-Vektor-Maschinen, bei denen es sich um einen weiteren Typ der linearen Diskriminanzfunktion handelt.

Die logistische Diskriminanz wird ausführlicher bei Anderson (1982) und McLachlan (1992) behandelt. Die logistische (sigmoidale) Funktion ist die Umkehrung der Logit-Funktion, welche als *kanonische Verknüpfung* im Fall von Bernoulli-Stichproben aufgefasst werden kann. Die Softmax-Funktion ist die Verallgemeinerung des Ganzen auf multinomiale Stichproben. Zusätzliche Informationen zu derlei *generalisierten linearen Modellen* finden sich bei McCullogh und Nelder (1989).

GENERALISIERTE LINEARE MODELLE

Im Zusammenhang mit Suchmaschinen, der Informationsrückgewinnung (Information Retrieval) und der natürlichen Sprachverarbeitung sind Ranking-Probleme in jüngerer Zeit zu einem Hauptanwendungsgebiet des maschinellen Lernens geworden. Einen umfassenden Überblick über Algorithmen des maschinellen Lernens und wichtige Anwendungen gibt Liu 2011. Das in diesem Kapitel betrachtete Modell ist linear; in Abschnitt 13.11 diskutieren wir ein Ranking-Verfahren, das Kernel-Maschinen verwendet, wobei wir ein nichtlineares Modell erhalten, das die Integration verschiedener Ähnlichkeitsmaße erlaubt.

Das Verallgemeinern linearer Modelle durch die Verwendung nichtlinearer Basisfunktionen ist eine sehr alte Idee. In Kapitel 11 diskutieren wir mehrlagige Perzeptronen und in Kapitel 12 radiale Basisfunktionen, wobei auch die Parameter der Basisfunktionen während des Lernens der Diskriminanzfunktion aus den Daten gelernt werden. Support-Vektor-Maschinen (Kapitel 13) verwenden Kernel-Funktionen, die aus solchen Basisfunktionen konstruiert sind.

10.11 Übungen

1. Beschreiben Sie für jede der folgenden Basisfunktionen, wo sie ungleich Null ist:

   a) $\sin(x_1)$

   b) $\exp(-(x_1 - a)^2/c)$

   c) $\exp(-\|\boldsymbol{x} - \boldsymbol{a}\|^2/c)$

   d) $\log(x_2)$

   e) $1(x_1 > c)$

   f) $1(ax_1 + bx_2 > c)$.

2. Beweisen Sie Gleichung 10.4 und 10.5 für den zweidimensionalen Fall aus Abbildung 10.2.

3. Zeigen Sie, dass die Ableitung der Softmax-Funktion,

$$y_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}\,, \tag{10.50}$$

   gleich $\partial y_i/\partial a_j = y_i(\delta_{ij} - y_j)$ ist, wobei δ_{ij} gleich 1 ist, falls $i = j$ und ansonsten 0.

4. Beweisen Sie für $K = 2$, dass die Verwendung zweier Softmax-Ausgaben zum selben Ergebnis führt wie die Verwendung einer Ausgabe der Sigmoidfunktion.

Lösung:

$$y_1 = \frac{\exp \sigma_1}{\exp \sigma_1 + \exp \sigma_2} = \frac{1}{1 + \exp(\sigma_2 - \sigma_1)}$$
$$= \frac{1}{1 + \exp(-(\sigma_1 - \sigma_2))}$$
$$= \text{sigmoid}(\sigma_1 - \sigma_2) \, .$$

Wenn wir zum Beispiel $\sigma_1 = \boldsymbol{w}_1^T \boldsymbol{x}$ haben, dann ergibt sich

$$y_1 = \frac{\exp \boldsymbol{w}_1^T \boldsymbol{x}}{\exp \boldsymbol{w}_1^T \boldsymbol{x} + \exp \boldsymbol{w}_2^T \boldsymbol{x}} = \text{sigmoid}(\boldsymbol{w}_1^T \boldsymbol{x} - \boldsymbol{w}_2^T \boldsymbol{x})$$
$$= \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x})$$

mit $\boldsymbol{w} \equiv \boldsymbol{w}_1 - \boldsymbol{w}_2$ und $y_2 = 1 - y_1$.

5. Was können wir aus $\mathbf{W}_i$ in Gleichung 10.34 erschließen?

   Lösung: Wenn wir zum Beispiel die beiden Eingaben x_1 und x_2 haben, dann erhalten wir

$$\log \frac{p(x_1, x_2|\mathcal{C}_i)}{p(x_1, x_2|\mathcal{C}_K)} = \mathbf{W}_{i11} x_1^2 + \mathbf{W}_{i12} x_1 x_2 + \mathbf{W}_{i21} x_2 x_1 + \mathbf{W}_{i22} x_2^2$$
$$+ w_{i1} x_1 + w_{i2} x_2 + w_{i0} \, .$$

   Wir können den Gradientenabstieg anwenden und nach einem beliebigen $\mathbf{W}_{jkl}$ ableiten, um eine Aktualisierungvorschrift zu berechnen:

$$\Delta \mathbf{W}_{jkl} = \eta \sum_t (r_j^t - y_j^t) x_k^t x_l^t \, .$$

6. Wenn wir wie in Gleichung 10.34 quadratische Diskriminanzfunktionen (oder Diskriminanzfunktionen höherer Ordnung) verwenden, wie können wir dann die Varianz unter Kontrolle behalten?

7. Was folgt daraus, wenn wir beim Gradientenabstieg für alle x_j ein einziges η verwenden?

   Lösung: Die Verwendung eines einzigen η für alle x_j bedeutet, dass die Aktualisierungen alle auf der gleichen Skala erfolgen, woraus wiederum folgt, dass alle x_j die gleiche Skala haben sollten. Wenn dies nicht der Fall sein sollte, ist es eine gute Idee, vor dem Training alle x_j zu normalisieren, beispielsweise durch eine z-Normalisierung. Dabei ist zu beachten, dass die Skalenparameter für alle Eingaben gespeichert werden müssen, damit die gleiche Skalierung später auch für die Testinstanzen durchgeführt werden kann.

8. Welche Bedeutung haben w und w_0 im univariaten Fall der Klassifikation wie in Abbildung 10.6?

   Lösung: Steigung und Höhe der Geraden

9. Angenommen, für ein univariates x gehören $x \in (2,4)$ zu $\mathcal{C}_1$ und $x < 2$ oder $x > 4$ gehören zu $\mathcal{C}_2$. Wie können wir die beiden Klassen durch eine lineare Diskriminanzfunktion separieren?

   LÖSUNG: Wir definieren eine zusätzliche Variable $z \equiv x^2$ und verwenden die lineare Diskriminanzfunktion $w_2 z + w_1 x + w_0$ im (z, x)-Raum. Dies entstspricht einer quadratischen Diskriminanzfunktion im x-Raum. Beispielsweise können wir schreiben

$$\text{wähle} \begin{cases} \mathcal{C}_1 & \text{falls } (x-3)^3 - 1 \leq 0 \\ \mathcal{C}_2 & \text{sonst} \end{cases}$$

   oder wir verwenden eine Sigmoidfunktion (siehe Abbildung 10.10):

$$\text{wähle} \begin{cases} \mathcal{C}_1 & \text{falls sigmoid}\big((x-3)^3 - 1\big) \leq 0{,}5 \\ \mathcal{C}_2 & \text{sonst .} \end{cases}$$

   Oder wir verwenden *zwei* lineare Diskriminanzfunktionen im x-Raum, eine bei 2 und eine bei 4, und kombinieren sie dann durch OR. Solche geschichteten linearen Diskriminanzfunktionen werden wir in Kapitel 11 behandeln.

10. Definieren Sie für die Beispieldaten aus Abbildung 10.9 eine Rangordnung, die ein lineares Modell nicht lernen könnte. Erläutern Sie, wie das Modell verallgemeinert werden kann, damit das Lernen möglich wird.

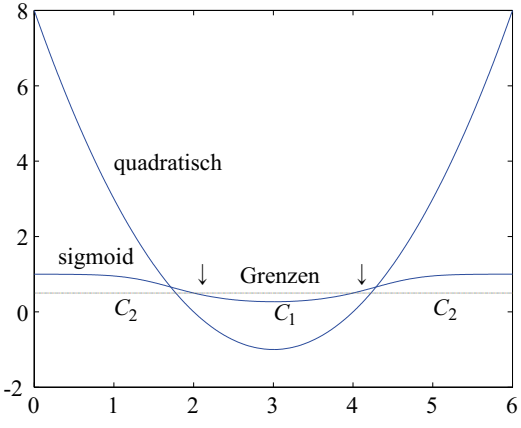


***Abb. 10.10*:** Die quadratische Diskriminanzfunktion vor und nach der Anwendung Sigmoidfunktion. Die Grenzen befinden sich dort, wo die Diskriminanzfunktion 0 oder die Sigmoidfunktion 0,5 ist.

10.12 Literaturangaben

Aizerman, M. A., E. M. Braverman, and L. I. Rozonoer. 1964. „Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning.“ *Automation and Remote Control* 25: 821–837.

Anderson, J. A. 1982. „Logistic Discrimination.“ In *Handbook of Statistics,* Vol. 2, *Classification, Pattern Recognition and Reduction of Dimensionality,* ed. P. R. Krishnaiah, L. N. Kanal, 169–191. Amsterdam: North Holland.

Bridle, J. S. 1990. „Probabilistic Interpretation of Feedforward Classification Network Outputs with Relationships to Statistical Pattern Recognition.“ In *Neurocomputing: Algorithms, Architectures and Applications,* ed. F. Fogelman-Soulie, J. Herault, 227–236. Berlin: Springer.

Burges, C., T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. 2005. „Learning to Rank using Gradient Descent.“ In *22nd International Conference on Machine Learning,* 89–96, New York: ACM Press.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification,* 2nd ed. New York: Wiley.

Liu, T.-Y. 2011. *Learning to Rank for Information Retrieval.* Heidelberg: Springer.

McCullagh, P., and J. A. Nelder. 1989. *Generalized Linear Models.* London: Chapman and Hall.

McLachlan, G. J. 1992. *Discriminant Analysis and Statistical Pattern Recognition.* New York: Wiley.

Shen, L., and A. K. Joshi. 2005. „Ranking and Reranking with Perceptron.“ *Machine Learning* 60: 73–96.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

11 Mehrlagige Perzeptronen

Das mehrlagige Perzeptron ist eine künstliche neuronale Netzstruktur und gleichzeitig ein nichtparametrischer Schätzer, der sowohl für die Klassifikation als auch die Regression verwendet werden kann. Wir diskutieren den Backpropagation-Algorithmus, mit dem ein mehrlagiges Perzeptron für eine Vielzahl von Anwendungen trainiert werden kann.

11.1 Einführung

Modelle künstlicher neuronaler Netze, zu denen das in diesem Kapitel diskutierte *Perzeptron* gehört, sind dem menschlichen Gehirn nachempfunden. Es gibt Kognitionswissenschaftler und Neurowissenschaftler, deren Ziel es ist, die Funktionsweise des Gehirns zu verstehen (Posner 1989; Thagard 2005) und die zu diesem Zweck Modelle der natürlichen neuronalen Netze im Gehirn erstellen und Simulationsstudien durchführen.

künstliche neuronale Netze

Die Ingenieurwissenschaften sehen ihr Ziel jedoch nicht darin, das Gehirn per se zu verstehen, sondern darin, nützliche Maschinen zu bauen. Uns interessieren *künstliche neuronale Netze*, weil wir davon überzeugt sind, dass sie uns helfen können, bessere Computersysteme zu erschaffen. Das Gehirn ist ein „Gerät“ zur Informationsverarbeitung, das über einige unglaubliche Fähigkeiten verfügt und gegenwärtige Ingenieurprodukte in vielen Bereichen deutlich übertrifft, beispielsweise im Sehvermögen, bei der Spracherkennung und beim Lernen, um nur drei zu nennen. Diese Anwendungen haben offensichtliche ökonomische Verwertbarkeit, wenn sie in Maschinen zur Verwendung kommen. Wenn wir verstehen können, wie das Gehirn diese Funktionen ausführt, so sind wir in der Lage, Lösungen für diese Aufgaben als formale Algorithmen zu definieren und sie dann auf Rechnern zu implementieren.

Neuronen

Das menschliche Gehirn unterscheidet sich doch deutlich von einem Computer. Während ein Computer im Allgemeinen einen Prozessor besitzt, besteht das Gehirn aus einer großen Anzahl (10^{11}) von Verarbeitungseinheiten, nämlich den parallel operierenden *Neuronen*. Zwar sind die Details nicht bekannt, jedoch glaubt man, dass die Verarbeitungseinheiten viel einfacher und langsamer arbeiten als ein Prozessor in einem Computer. Was das Gehirn auch unterscheidet und allem Anschein nach für dessen Rechenleistung verantwortlich ist, ist die große innere Konnektivität: Neu-

Synapsen

ronen im Gehirn sind über sogenannte *Synapsen* mit etwa 10^4 anderen Neuronen verbunden, die alle parallel arbeiten. In einem Computer ist der Prozessor aktiv und die separate Speichereinheit passiv. Beim Gehirn aber wird davon ausgegangen, dass sowohl die Verarbeitung als auch die Speicherung gemeinsam auf das Netz verteilt werden; erstere wird durch die Neuronen vorgenommen und letztere findet in den Synapsen zwischen den Neuronen statt.

11.1.1 Das Gehirn verstehen

Analyseebenen

Laut Marr (1982) findet das Verständnis eines Informationsverarbeitungssystems auf drei Ebenen statt, den sogenannten *Analyseebenen*:

1. Die *Rechentheorie* entspricht dem Ziel der Berechnung und einer abstrakten Definition der Aufgabenstellung.
2. Mit *Repräsentation und Algorithmus* meinen wir die Darstellung der Eingabe und Ausgabe sowie die Spezifikation des Algorithmus für die Transformation von Eingabe zu Ausgabe.
3. Die *Hardware-Implementierung* ist die eigentliche physische Realisierung des Systems.

Ein Beispiel findet sich in der Sortierung. Die Rechentheorie besteht darin, eine vorliegende Menge an Elementen zu ordnen. Die Repräsentation kann Ganzzahlen verwenden und als Algorithmus könnte Quicksort gewählt werden. Nach dem Kompilieren ist der ausführbare Code für einen speziellen Prozessor, der binär dargestellte Ganzzahlwerte sortiert, eine Hardware-Implementierung.

Die Idee ist die, dass für dieselbe Rechentheorie multiple Darstellungen und Algorithmen zur Manipulation der Symbole in dieser Darstellung existieren können. Äquivalent dazu können auch für eine beliebige Repräsentation und einen Algorithmus verschiedene Hardware-Implementierungen möglich sein. Wir können einen der zahlreichen Sortieralgorithmen wählen, und selbst der gleiche Algorithmus kann auf Rechnern mit verschiedenen Prozessoren kompiliert werden und zu verschiedenen Hardware-Implementierungen führen.

Als anderes Beispiel seien „6“, „VI“ und „110“ genannt – drei verschiedene Darstellungen der Zahl Sechs. Es existieren verschiedenen Algorithmen für Additionen, je nachdem, welche Repräsentation verwendet wird. Digitale Rechner nutzen die binäre Repräsentation und verfügen über einen Kreislauf, um bei dieser Darstellungsweise zu addieren, was einer speziellen Hardware-Implementierung gleich kommt. Die für uns gewöhnlichen Zahlen werden auf andere Weise dargestellt und ihre Addition entspricht einer anderen Menge an Anweisungen für einen Abakus, was eine weitere Hardware-Implementierung darstellt. Addieren wir zwei Zahlen in

unserem Gehirn, so wird noch eine dritte Repräsentation und ein ihr angemessener Algorithmus genutzt, welcher durch die Neuronen implementiert wird. Doch all diese verschiedenen Hardware-Implementierungen – beispielsweise wir selbst, der Abakus, ein digitaler Rechner – setzen dieselbe Rechentheorie um, nämlich die Addition.

Das klassische Beispiel bildet der Unterschied zwischen natürlichen und künstlichen Flugmaschinen. Eine Schwalbe schlägt ihre Flügel; ein Passagierflugzeug tut dies jedoch nicht, sondern nutzt Düsentriebwerke. Die Schwalbe und das Flugzeug stellen zwei Hardware-Implementierungen dar, die für unterschiedliche Zwecke entworfen wurden und unterschiedlichen Einschränkungen unterliegen. Sie implementieren jedoch beide dieselbe Theorie, und zwar die der Aerodynamik.

Das Gehirn ist eine Hardware-Implementierung zum Lernen oder zur Mustererkennung. Wenn es uns möglich ist, anhand dieser speziellen Implementierung mittels Reverse Engineering, also quasi einem Nachbau, die verwendete Darstellung und den Algorithmus zu extrahieren, dann können wir daraus möglicherweise die Rechentheorie ableiten. Dann wiederum können wir andere Darstellungen und Algorithmen nutzen, um auf dieser Grundlage eine Hardware-Implementierung zu erstellen, die besser auf die Mittel und Einschränkungen, denen wir unterliegen, zugeschnitten ist. Die Hoffnung ist die, dass unsere Implementierung kostengünstiger, schneller und genauer ist.

Genau wie die ersten Versuche, Flugmaschinen zu bauen, die Vögeln doch sehr ähnlich sahen – bis wir die Gesetze der Aerodynamik entdeckten – so ist auch zu erwarten, dass die ersten Ansätze für die Konstruktion von Strukturen mit Fähigkeiten ähnlich denen des Gehirns ebenfalls dem Gehirn ähneln werden, mit Netzen aus einer großen Zahl an Verarbeitungseinheiten. Erst wenn wir schließlich die Rechentheorie hinter dem Konzept der Intelligenz begreifen, wird sich dies unter Umständen ändern. Es kann also festgestellt werden, dass wir uns beim Verständnis des Gehirns, wenn wir an künstlichen neuronalen Netzen arbeiten, auf der Ebene der Repräsentation und der Algorithmen bewegen.

Ebenso wie Federn irrelevant für das Fliegen sind, werden wir im Laufe der Zeit möglicherweise entdecken, dass Neuronen und Synapsen für die Intelligenz unbedeutend sind. Bis dahin jedoch gibt es noch einen weiteren Grund, warum wir versuchen, die Funktionsweise des Gehirns zu verstehen, und dieser hat mit der parallelen Verarbeitung zu tun.

11.1.2 Neuronale Netze als Paradigma für die Parallelverarbeitung

Seit den 1980ern sind Computersysteme mit Tausenden von Prozessoren kommerziell verfügbar. Die Software für solche parallelen Architekturen hat sich allerdings nicht so rasant weiterentwickelt wie die Hardware. Die

Ursache dafür liegt darin, dass fast unsere gesamte Rechentheorie bis zu dem Zeitpunkt auf seriellen Einprozessormaschinen basierte. Wir sind nicht in der Lage, die uns verfügbaren parallelen Maschinen effizient zu nutzen, weil wir sie nicht effizient programmieren können.

Parallel-verarbeitung

Es existieren hauptsächlich zwei Paradigmen für die *Parallelverarbeitung*. Bei den sogenannten SIMD-Maschinen (Single Instruction Multiple Data) führen alle Prozessoren dieselbe Anweisung aus, jedoch auf unterschiedlichen Daten. Bei den MIMD-Maschinen (Multi Instruction Multiple Data) können verschiedene Prozessoren unterschiedliche Anweisungen auf unterschiedlichen Daten ausführen. SIMD-Maschinen sind einfacher zu programmieren, da nur ein Programm zu schreiben ist. Allerdings haben Problemstellungen nur selten eine so regelmäßige Struktur, dass sie mittels einer SIMD-Maschine parallelisiert werden können. MIMD-Maschinen sind allgemeiner gehalten, bringen jedoch größere Schwierigkeiten mit sich, separate Programme für all die individuellen Prozessoren zu schreiben; zusätzliche Probleme entstehen hinsichtlich der Synchronisation, dem Datentransfer zwischen Prozessoren, und so weiter. SIMD-Maschinen sind einfacher zu konstruieren und Maschinen mit mehr Prozessoren können gebaut werden, wenn sie auf dem SIMD-Prinzip basieren. Bei MIMD-Maschinen sind die Prozessoren komplexer und ein aufwendigeres Kommunikationsnetzwerk sollte für den willkürlichen Datenaustausch zwischen den Prozessoren erschaffen werden.

Nehmen wir nun an, dass wir Maschinen haben, deren Prozessoren ein klein wenig komplexer sind als die SIMD-Prozessoren, jedoch nicht so komplex wie MIMD-Prozessoren. Nehmen wir weiter an, uns liegen einfache Prozessoren mit einem kleinen lokalen Speicher vor, in dem einige Parameter abgelegt werden können. Jeder Prozessor implementiert eine feste Funktion und führt dieselbe Anweisung wie SIMD-Prozessoren aus; indem wir aber verschiedene Werte in die lokalen Speicher laden, können sie unterschiedliche Dinge tun und die gesamte Operation kann auf solche Prozessoren verteilt werden. Wir erhalten dann sogenannte NIMD-Maschinen (Neural Instruction Multiple Data), bei denen jeder Prozessor einem Neuron gleichkommt, lokale Parameter den synaptischen Gewichten entsprechen und die gesamte Struktur ein neuronales Netz bildet. Ist die in jedem Prozessor implementierte Funktion einfach und der lokale Speicher klein, so passen zahlreiche solche Prozessoren auf einen einzelnen Chip.

Das Problem besteht nun darin, eine Aufgabe auf ein Netz solcher Prozessoren zu verteilen und die lokalen Parameterwerte zu bestimmen. An dieser Stelle kommt das Lernen ins Spiel: wir müssen solche Maschinen nicht programmieren und die Parameterwerte selbst bestimmen, wenn solche Maschinen anhand von Beispielen lernen können.

Somit stellen künstliche neuronale Netze eine Möglichkeit dar, die parallele Hardware, die wir dank heutiger Technologie zur Verfügung haben, zu nutzen und dank des Lernens müssen wir sie nicht einmal program-

mieren. Wir selbst sind also von der Last, sie programmieren zu müssen, entbunden.

In diesem Kapitel diskutieren wir derlei Strukturen und wie sie *trainiert* werden. Es sei daran erinnert, dass die Operation eines künstlichen neuronalen Netzes eine mathematische Funktion ist, welche auf einem seriellen Rechner implementiert werden kann – was normalerweise auch der Fall ist – und dass sich das *Training* des Netzes kaum von den statistischen Techniken unterscheidet, die wir in den vergangen Kapiteln besprochen haben. Die Vorstellung, dass diese Operation auf einem Netz einfacher Verarbeitungseinheiten ausgeführt wird, ist nur dann sinnvoll, wenn wir die parallele Hardware zur Verfügung haben und wenn das Netz so groß ist, dass es nicht schnell genug auf einem seriellen Rechner simuliert werden kann.

11.2 Das Perzeptron

PERZEPTRON

VERBINDUNGSGEWICHTUNG

SYNAPTISCHES GEWICHT

Das *Perzeptron* ist das grundlegende Verarbeitungselement. Es nimmt Eingaben entgegen, die entweder der Umgebung entstammen oder die Ausgaben anderer Perzeptronen sind. Mit jeder Eingabe $x_j \in \mathbb{R} <$, $j = 1, \ldots, d$ ist ein *Verbindungsgewicht* oder auch *synaptisches Gewicht* $w_j \in \mathbb{R}$ assoziiert und die Ausgabe y ist im einfachsten Fall eine gewichtete Summe der Eingaben (siehe Abbildung 11.1):

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \,. \tag{11.1}$$

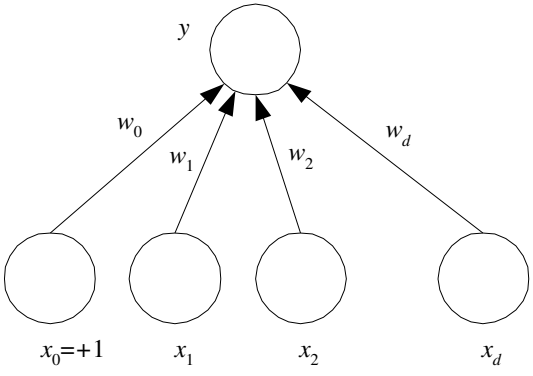


Abb. 11.1: Ein einfaches Perzeptron. $x_j, j = 1, \ldots, d$ sind die Eingabeeinheiten. x_0 ist die Eingae für die Verzerrung mit dem immer gleichen Wert 1. y ist die Ausgabeeinheit. w_j ist das Gewicht der direkten Verbindung von Eingabe x_j zur Ausgabe.

w_0 ist der Wert des Ordinatenabschnitts, um das Modell allgemeiner zu halten; im Allgemeinen ist er als Gewicht für die Eingabe x_0, die immer gleich +1 ist, modelliert. Wir können die Ausgabe des Perzeptrons als ein Skalarprodukt

$$y = \boldsymbol{w}^T \boldsymbol{x} \tag{11.2}$$

schreiben, wobei $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$ und $\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$ *erweiterte* Vektoren sind, um das Gewicht und die Eingabe der Verzerrung einzubeziehen.

VERZERRUNGS-EINHEIT

Während der Testphase berechnen wir bei gegebenen Gewichten $\boldsymbol{w}$ für die Eingabe $\boldsymbol{x}$ die Ausgabe y. Um eine vorliegende Aufgabenstellung zu implementieren, müssen wir die Gewichte $\boldsymbol{w}$, also die Parameter des Systems, *trainieren*, damit korrekte Ausgaben für die gegebenen Eingaben generiert werden.

Ist $d = 1$ und wird x aus der Umgebung durch eine Eingabeeinheit eingespeist, so erhalten wir

$$y = wx + w_0 \,,$$

was der Gleichung für eine Gerade mit Anstieg w und der Ordinate w_0 des Ordinatenschnittpunkts entspricht. Somit kann dieses Perzeptron mit einer Eingabe und einer Ausgabe genutzt werden, um eine lineare Anpassung zu implementieren. Mit mehr als einer Eingabe wird die Gerade zur (Hyper-) Ebene und das Perzeptron mit mehr als einer Eingabe kann verwendet werden, um eine multivariate lineare Anpassung zu implementieren. Bei vorliegender Stichprobe können die Parameter w_j mittels Regression herausgefunden werden (siehe Abschnitt 5.8).

Das in Gleichung 11.1 definierte Perzeptron beschreibt eine Hyperebene und kann als solches genutzt werden, um den Eingaberaum in zwei Teile zu zerlegen: den positiven und den negativen Halbraum (siehe Kapitel 10). Wenn wir es verwenden, um eine lineare Diskriminanzfunktion zu implementieren, kann das Perzeptron zwei Klassen trennen, indem wir das Vorzeichen der Ausgabe überprüfen. Wenn wir $s(\cdot)$ als die *Schwellwertfunktion*

SCHWELLWERT-FUNKTION

$$s(a) = \begin{cases} 1 \text{ falls } a > 0 \,, \\ 0 \text{ andernfalls} \end{cases} \tag{11.3}$$

definieren, so erhalten wir

$$\text{wähle} \begin{cases} \mathcal{C}_1 \text{ falls } s(\boldsymbol{w}^T \boldsymbol{x}) > 0 \,, \\ \mathcal{C}_2 \text{ andernfalls} \,. \end{cases}$$

Es sei daran erinnert, dass die Verwendung einer linearen Diskriminante auf der Vermutung basiert, dass Klassen linear trennbar sind. Das heißt, es

wird angenommen, dass eine Hyperebene $\boldsymbol{w}^T\boldsymbol{x} = 0$ gefunden werden kann, die $\boldsymbol{x}^t \in \mathcal{C}_1$ und $\boldsymbol{x}^t \in \mathcal{C}_2$ voneinander trennt. Wenn wir zu einem späteren Zeitpunkt die a-posteriori-Wahrscheinlichkeit benötigen – beispielsweise um das Risiko zu kalkulieren – müssen wir die Sigmoidfunktion auf die Ausgabe anwenden

$$\begin{aligned} o &= \boldsymbol{w}^T\boldsymbol{x}\,, \\ y &= \text{sigmoid}(o) = \frac{1}{1+\exp[-\boldsymbol{w}^T\boldsymbol{x}]}\,. \end{aligned} \tag{11.4}$$

Wenn es $K > 2$ Ausgaben gibt, so existieren K Perzeptronen, von denen jedes einen Gewichtsvektor $\boldsymbol{w}_i$ besitzt (siehe Abbildung 11.2):

$$\begin{aligned} y_i &= \sum_{j=1}^{d} w_{ij}x_j + w_{i0} = \boldsymbol{w}_i^T\boldsymbol{x}\,, \\ \boldsymbol{y} &= \mathbf{W}\boldsymbol{x} \end{aligned} \tag{11.5}$$

mit w_{ij} als Gewicht von Eingabe x_j zu Ausgabe y_i. $\mathbf{W}$ ist die $K \times (d+1)$ Gewichtsmatrix von w_{ij}, deren Zeilen den Gewichtsvektoren der K Perzeptronen entsprechen. Bei der Nutzung im Rahmen der Klassifikation gilt während der Testphase:

$$\text{wähle } \mathcal{C}_i, \text{ falls } y_i = \max_k y_k\,.$$

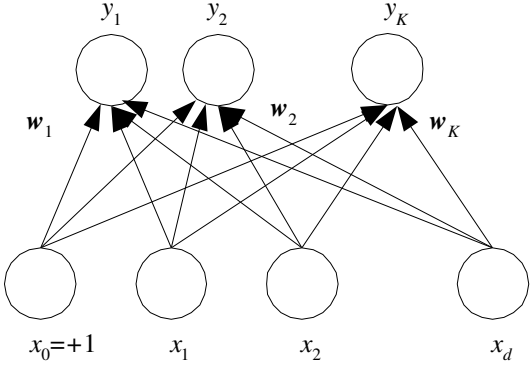


Abb. 11.2: K parallele Perzeptronen. $x_j, j = 0, \ldots, d$ sind die Eingaben und $y_i, i = 1, \ldots, K$ sind die Ausgaben. w_{ij} ist das Gewicht der Verbindung von Eingabe x_j zu Ausgabe y_i. Jede Ausgabe ist eine gewichtete Summe der Eingaben. Bei der Verwendung für Klassifikationsprobleme mit K Klassen kann in einem Nachbearbeitungsschritt das Maximum gewählt oder die Softmax-Funktion verwendet werden, wenn wir die a-posteriori-Wahrscheinlichkeiten benötigen.

Jedes Perzeptron ist eine *lokale* Funktion seiner Eingaben und synaptischen Gewichte. Wenn wir bei der Klassifikation die a-posteriori-Wahrscheinlichkeiten (statt nur des Codes der Gewinnerklasse) benötigen und die Softmax-Funktion verwenden, dann benötigen wir die Werte aller Ausgaben. Die Implementierung durch ein neuronales Netz führt auf einen Zweiphasenprozess, bei dem in der ersten Phase die gewichteten Summen und in der zweiten Phase die Softmax-Werte berechnet werden. Wir schreiben dies aber weiterhin als einzelne Schicht:

$$\begin{aligned} o_i &= \boldsymbol{w}_i^T \boldsymbol{x} \,, \\ y_i &= \frac{\exp o_i}{\sum_k \exp o_k} \,. \end{aligned} \tag{11.6}$$

Es sei daran erinnert, dass durch die Definition von Hilfseingaben das lineare Perzeptron auch für polynomiale Approximationen verwendet werden kann. Beispielsweise sei definiert, dass $x_3 = x_1^2, x_4 = x_2^2, x_5 = x_1 x_2$ (Abschnitt 10.2). Das Gleiche kann auch bei Perzeptronen angewandt werden (Durbin und Rumelhart 1989). In Abschnitt 11.5 werden wir mehrlagige Perzeptronen betrachten, bei denen solche nichtlinearen Funktionen anhand von Daten in einer verborgenen Schicht gelernt werden, statt sie a priori vorauszusetzen.

Jede der in Kapitel 10 diskutierten Methoden zur linearen Diskriminanzanalyse kann verwendet werden, um $\boldsymbol{w}_i, i = 1, \ldots, K$ offline zu berechnen und dann ins Netz einzusetzen. Dies beinhaltet auch den parametrischen Ansatz mit einer gemeinsamen Kovarianzmatrix, die logistische Diskriminanz, die Diskriminanz durch Regression und Support-Vektor-Maschinen. In einigen Fällen steht uns nicht die vollständige Stichprobe zur Verfügung wenn die Trainingsphase einsetzt, und wir müssen die Parameter iterativ aktualisieren, sobald neue Beispiele eintreffen; wir werden diesen Fall des *online*-Lernens in Abschnitt 11.3 behandeln.

Gleichung 11.5 definiert eine lineare Transformation von einem d-dimensionalen Raum in einen K-dimensionalen Raum und kann ebenfalls für die Dimensionalitätsreduktion verwendet werden, falls $K < d$. Jede der Methoden aus Kapitel 6 kann angewandt werden, um $\mathbf{W}$ offline zu berechnen und dann die Perzeptronen zu nutzen, um die Transformation zu implementieren, zum Beispiel die PCA. In so einem Fall haben wir ein zweilagiges Netz, bei dem die erste Schicht an Perzeptronen die lineare Transformation und die zweite Schicht die lineare Regression oder Klassifikation im neuen Raum implementiert. Da es sich bei beiden um lineare Transformationen handelt, stellen wir fest, dass sie kombiniert und als eine einzelne Schicht geschrieben werden können. Den interessanteren Fall, bei dem die erste Schicht eine *nichtlineare* Dimensionalitätsreduktion implementiert, werden wir in Abschnitt 11.5 betrachten.

11.3 Training eines Perzeptrons

Das Perzeptron definiert eine Hyperebene, und das Perzeptron eines neuronalen Netzes ist lediglich eine Möglichkeit, die Hyperebene zu *implementieren*. Bei vorliegender Stichprobe können die Gewichtswerte *offline* berechnet werden. Setzt man sie dann ein, kann das Perzeptron genutzt werden, um die Ausgabewerte zu errechnen.

Beim Training neuronaler Netze wenden wir im Allgemeinen das online-Lernen an, bei dem uns nicht die komplette Stichprobe vorliegt, sondern wir eine Instanz nach der anderen zu sehen bekommen und dabei vom Netz eine Aktualisierung seiner Parameter nach jeder Instanz erwarten, so dass es sich also selbst über die Zeit hinweg anpasst. Solch ein Ansatz ist aus verschiedenen Gründen interessant:

1. Er erspart es uns, die Stichprobe in einer externen Speichereinheit ablegen zu müssen und die Zwischenergebnisse während der Optimierung zu speichern. Ein Ansatz wie die Support-Vektor-Maschinen (Kapitel 13) kann sich bei großen Stichproben als recht kostspielig herausstellen, und bei einigen Anwendungen ziehen wir unter Umständen eine einfachere Vorgehensweise vor, bei der wir nicht die gesamte Stichprobe speichern müssen, um an ihr ein komplexes Optimierungsproblem zu lösen.
2. Das Problem könnte sich im Laufe der Zeit verändern, was bedeutet, dass die Stichprobenverteilung keine feste ist und ein Datensatz nicht a priori ausgewählt werden kann. Zum Beispiel könnten wir ein Spracherkennungssystem implementieren, welches sich selbst an den Nutzer anpasst.
3. Es könnten physische Veränderungen im System auftreten. Beispielsweise nutzen sich die Komponenten eines robotronischen Systems im Laufe der Zeit ab, oder Sensoren könnten ihre Leistungsfähigkeit verlieren.

Online-Lernen

Beim *online-Lernen* schreiben wir die Fehlerfunktion nicht für die komplette Stichprobe, sondern für individuelle Instanzen. Wir beginnen mit zufälligen Gewichten und passen bei jeder Iteration die Parameter ein wenig mehr an, um den Fehler zu minimieren, ohne dabei das zu vergessen, was wir bereits erlernt haben. Wenn diese Fehlerfunktion ableitbar ist, dann können wir den Gradientenabstieg verwenden.

Zum Beispiel ist bei der Regression der Fehler am einzelnen Instanzpaar mit Index t, (x^t, r^t), gleich

$$E^t(\boldsymbol{w}|\boldsymbol{x}^t, r^t) = \frac{1}{2}(r^t - y^t)^2 = \frac{1}{2}[r^t - (\boldsymbol{w}^T\boldsymbol{x}^t)]^2$$

und für $j = 0, \ldots, d$ ist die online-Aktualisierung gleich

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t\,, \tag{11.7}$$

STOCHASTISCHER GRADIENTEN-ABSTIEG

wobei η dem Lernfaktor entspricht, welcher schrittweise hin zur Konvergenz abnimmt. Dies ist als *stochastischer Gradientenabstieg* bekannt.

Ähnlich dazu können Aktualisierungsregeln für Klassifikationsprobleme abgeleitet werden, indem die logistische Diskriminanz verwendet wird, bei der Aktualisierungen nach jedem Muster vorgenommen werden, statt sie zu summieren und die Aktualisierung nach einem kompletten Durchlauf durch die *Trainingsmenge* einzubringen. Für eine einzelne Instanz $(\boldsymbol{x}^t, \boldsymbol{r}^t)$ mit $r_i^t = 1$, falls $\boldsymbol{x}^t \in \mathcal{C}_1$, und $r_i^t = 0$, falls $\boldsymbol{x}^t \in \mathcal{C}_2$, ist die einzelne Ausgabe bei zwei Klassen gleich

$$y^t = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x}^t)$$

und die Kreuzentropie ergibt sich als

$$E^t(\{\boldsymbol{w}\}|\boldsymbol{x}^t, \boldsymbol{r}^t) = -r^t \log y^t + (1 - r^t) \log(1 - y^t) \,.$$

Unter Verwendung des Gradientenabstiegs erhalten wir die folgende online-Aktualisierungsregel $j = 0, \ldots, d$:

$$\Delta w_j^t = \eta(r^t - y^t) x_j^t \,. \tag{11.8}$$

Wenn $K > 2$ Klassen vorliegen, ergeben sich die Ausgaben für die einzelne Instanz $(\boldsymbol{x}^t, \boldsymbol{r}^t)$ mit $r_i^t = 1$, falls $\boldsymbol{x}^t \in \mathcal{C}_i$, und $r_i^t = 0$ in allen anderen Fällen aus

$$y_i^t = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}^t}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}^t}$$

und die Kreuzentropie ist

$$E^t(\{\boldsymbol{w}_i\}_i|\boldsymbol{x}^t, \boldsymbol{r}^t) = -\sum_i r_i^t \log y_i^t \,.$$

Nutzen wir den Gradientenabstieg, so ergibt sich die folgende Regel für online-Aktualisierungen für $i = 1, \ldots, K,\ j = 0, \ldots, d$:

$$\Delta w_{ij}^t = \eta(r_i^t - y_i^t) x_j^t \,, \tag{11.9}$$

was den Gleichungen entspricht, die wir in Abschnitt 10.7 betrachtet haben, außer dass wir nicht über alle Instanzen hinweg summieren, sondern nach einer einzelnen Instanz aktualisieren. Der Pseudocode des Algorithmus ist im Listing 11.1 dargestellt; er entspricht der online-Version vom Listing 10.2.

Sowohl Gleichung 11.7 als auch 11.9 haben die Form

$$\begin{aligned} &\text{Aktualisierung} = \\ &\quad \text{Lernfaktor} \cdot (\text{GewünschteAusgabe} - \text{TatsächlicheAusgabe}) \cdot \text{Eingabe} \,. \end{aligned} \tag{11.10}$$

```
For i = 1, ..., K
    For j = 0, ..., d
        w_ij ← rand(−0,01, 0,01)
Repeat
    For alle (x^t, r^t) ∈ 𝒳 in zufälliger Reihenfolge
        For i = 1, ..., K
            o_i ← 0
            For j = 0, ..., d
                o_i ← o_i + w_ij x_j^t
        For i = 1, ..., K
            y_i ← exp(o_i) / Σ_k exp(o_k)
        For i = 1, ..., K
            For j = 0, ..., d
                w_ij ← w_ij + η(r_i^t − y_i) x_j^t
Until Konvergenz
```

***Listing 11.1*:** Trainingsalgorithmus für ein Perzeptron, welcher den stochastischen online-Gradientenabstieg für den Fall mit $K > 2$ Klassen implementiert. Hierbei handelt es sich um die online-Version des Algorithmus aus Abbildung 10.2.

Versuchen wir einmal zu verstehen, was hierbei passiert. Folgendes gilt: falls die tatsächliche Ausgabe gleich der gewünschten Ausgabe ist, wird keine Aktualisierung ausgelöst. Wird eine Aktualisierung vorgenommen, so erhöht sich ihre Größe, je mehr sich die Differenz zwischen der gewünschten Ausgabe und der tatsächlichen Ausgabe erhöht. Wir erkennen außerdem für den Fall, dass die tatsächliche Ausgabe geringer ist als die gewünschte Ausgabe, dass die Aktualisierung positiv ist, falls die Eingabe positiv ist und negativ, wenn die Eingabe negativ ist. Dies hat den Effekt, dass die tatsächliche Ausgabe erhöht und die Differenz verringert wird. Ist die tatsächliche Ausgabe größer als die gewünschte Ausgabe, so ist die Aktualisierung negativ, wenn die Eingabe positiv ist und positiv, wenn die Eingabe negativ ist; dadurch wird die tatsächliche Ausgabe verringert und näher an die gewünschte Ausgabe gebracht.

Wird eine Aktualisierung durchgeführt, dann hängt ihre Größe auch von der Eingabe ab. Wenn die Eingabe nahe 0 liegt, so ist ihr Effekt auf die tatsächliche Ausgabe gering und daher wird ihr Gewicht auch nur um einen kleinen Betrag aktualisiert. Je größer eine Eingabe, desto größer die Aktualisierung ihres Gewichts.

Und schließlich hängt die Größenordnung einer Aktualisierung auch vom Lernfaktor η ab. Ist er zu groß, so hängen Aktualisierungen zu stark von den aktuelleren Instanzen ab; es ist, als ob das System nur über ein Kurzzeitgedächtnis verfügt. Ist dieser Faktor klein, werden eventuell viele Aktualisierungen zur Konvergenz benötigt. In Abschnitt 11.8.1 diskutieren wir Methoden zum schnelleren Herbeiführen von Konvergenz.

11.4 Lernen von Booleschen Funktionen

Bei einer Booleschen Funktion sind die Eingaben binär und die Ausgabe ist gleich 1, wenn der entsprechende Funktionswert wahr ist, und in allen anderen Fällen gleich 0. Somit kann man hier von einem Klassifikationsproblem mit zwei Klassen sprechen. Ein Beispiel hierfür ist in Tabelle 11.1 gegeben; es zeigt die Eingaben und nötigen Ausgaben für das Erlernen der Verbindung zweier Instanzen durch die AND-Funktion.

Tabelle 11.1: Eingabe und Ausgabe für die AND-Funktion.

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

Ein Beispiel eines Perzeptrons, welches AND implementiert, und seine geometrische Interpretation in zwei Dimensionen ist in Abbildung 11.3 dargestellt. Die Diskriminante ist

$$y = s(x_1 + x_2 - 1{,}5) \,,$$

sprich, $\boldsymbol{x} = [1, x_1, x_2]^T$ und $\boldsymbol{w} = [-1{,}5, 1, 1]^T$. Man beachte, dass $y = x_1 + x_2 - 1{,}5$ die vier Einschränkungen beachtet, die durch die Definition der AND-Funktion in Tabelle 11.1 gegeben sind, beispielsweise für $x_1 = 1, x_2 = 0$, $y = s(-0{,}5) = 0$. Ähnlich dazu kann gezeigt werden, dass $y = s(x_1 + x_2 - 0{,}5)$ die OR-Funktion implementiert.

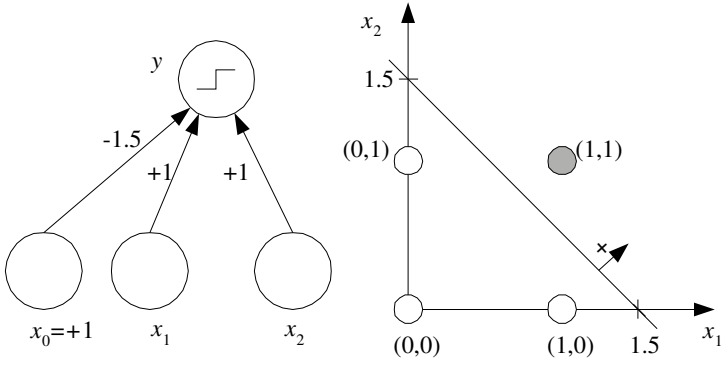


Abb. 11.3: Das Perzeptron, welches die AND-Funktion implementiert, und seine geometrische Interpretation.

Tabelle 11.2: Eingabe und Ausgabe für die XOR-Funktion.

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

Obwohl Boolesche Funktionen wie AND und OR linear trennbar sind und mit Hilfe des Perzeptrons gelöst werden können, trifft dies auf gewisse Funktionen wie XOR nicht zu. Die Auflistung von Eingaben und nötigen Ausgaben für XOR findet sich in Tabelle 11.2.

Wie wir in Abbildung 11.4 sehen, ist das Problem nicht linear trennbar. Das lässt sich auch durch die Feststellung beweisen, dass es keine Werte w_0, w_1 und w_2 gibt, die den folgenden Satz von Ungleichungen erfüllen:

$$\begin{aligned} w_0 &\leq 0\,, \\ w_2 + w_0 &> 0\,, \\ w_1 + w_0 &> 0\,, \\ w_1 + w_2 + w_0 &\leq 0\,. \end{aligned}$$

Das Ergebnis sollte uns nicht überraschen, da die VC-Dimension einer Geraden (in zwei Dimensionen) gleich drei ist. Bei zwei binären Eingaben gibt es vier Fälle, und somit wissen wir, dass Probleme mit zwei Eingaben existieren, die nicht mit einer Geraden lösbar sind; XOR ist eins davon.

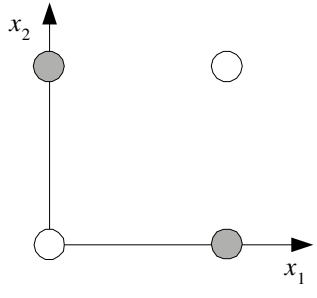


Abb. 11.4: Das XOR-Problem ist nicht linear trennbar. Wir können keine Gerade zeichnen, durch welche die leeren Kreise auf der einen und die schattierten Kreise auf der anderen Seite liegen.

11.5 Mehrlagige Perzeptronen

Ein Perzeptron, welches eine einzelne Schicht von Gewichten besitzt, kann nur lineare Funktionen der Eingabe approximieren und Probleme wie XOR nicht lösen, bei denen die zu schätzende Diskriminante nicht linear ist. Ähnlich dazu kann ein Perzeptron nicht bei der nichtlinearen Regression angewandt werden. Diese Einschränkung trifft nicht auf sogenannte

VERBORGENE SCHICHTEN
MEHRLAGIGE PERZEPTRONEN

Feedforward-Netze mit Zwischenschichten oder *verborgenen Schichten* zwischen den Eingabe- und Ausgabeschichten zu. Wenn solche *mehrlagigen Perzeptronen* (MLP) für die Klassifikation verwendet werden, können sie nichtlineare Diskriminanten implementieren, und bei der Verwendung im Rahmen von Regressionen können sie nichtlineare Funktionen der Eingabe approximieren.

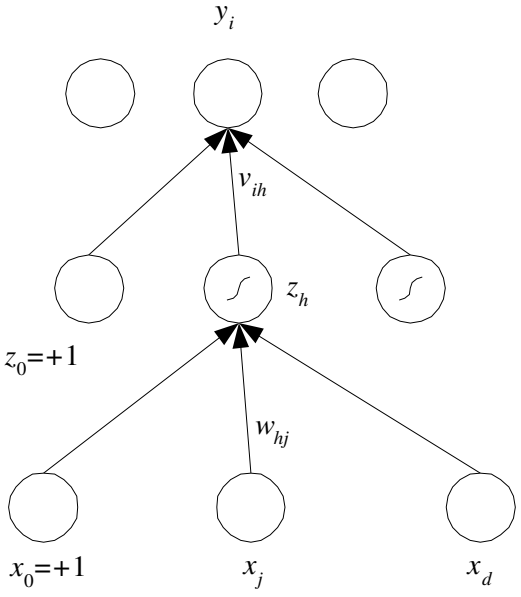


Abb. 11.5: Die Struktur eines mehrlagigen Perzeptrons. $x_j, j = 0, \ldots, d$ sind die Eingaben, und $z_h, h = 1, \ldots, H$ sind die verborgenen Einheiten, wobei H der Dimensionalität dieses verborgenen Raumes entspricht. z_0 ist die Verzerrung der verborgenen Schicht. $y_i, i = 1, \ldots, K$ sind die Ausgabeeinheiten. w_{hj} repräsentieren die Gewichte in der ersten Schicht und v_{ih} sind die Gewichte in der zweiten Schicht.

Die Eingabe $\boldsymbol{x}$ wird in die Eingabeschicht eingespeist (inklusive der Verzerrung), die „Aktivierung“ sorgt für eine Vorwärtspropagierung, und die Werte der verborgenen Einheiten z_h werden berechnet (siehe Abbildung 11.5). Jede verborgene Einheit ist ein Perzeptron für sich und wendet die nichtlineare Sigmoidfunktion auf seine gewichtete Summe an:

$$\begin{aligned} z_h &= \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}) \\ &= \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}, \quad h = 1, \ldots, H\,. \end{aligned} \tag{11.11}$$

Die Ausgabe y_i besteht aus Perzeptronen in der zweiten Schicht, welche die Ausgaben der verborgenen Einheiten als ihre Eingaben verwenden:

$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}\,, \tag{11.12}$$

wobei es auch eine Verzerrungseinheit in der verborgenen Schicht gibt, welche wir mit z_0 bezeichen; v_{i0} sind die Verzerrungsgewichtungen. Die

Eingabeschicht von x_j wird nicht mitgezählt, da dort keinerlei Berechnungen stattfinden, und wenn eine verborgene Schicht existiert, dann handelt es sich um ein zweilagiges Netz.

Wie üblich bei Regressionsproblemen existiert keine Nichtlinearität in der Ausgabeschicht, wenn y berechnet wird. Bei Problemen der Zweiklassendiskriminanz gibt es eine sigmoidale Ausgabeeinheit, und wenn $K > 2$ Klassen existieren, gibt es K Ausgaben mit Softmax als Nichtlinearität in der Ausgabe.

Wenn die Ausgaben der verborgenen Einheiten linear wären, so würde die verborgene Schicht keinen Nutzen haben: die lineare Kombination von linearen Kombinationen ergibt eine weitere lineare Kombination. Die Sigmoidfunktion ist die kontinuierliche, differenzierbare Version der Schwellwertbildung. Wir benötigen Differenzierbarkeit, weil die Lerngleichungen, die wir betrachten werden, gradientenbasiert sind. Eine andere sigmoidale (S-geformte) nichtlineare Basisfunktion, die verwendet werden kann, ist die hyperbolische Tangentenfunktion, tanh, welche zwischen -1 und $+1$ liegt, statt zwischen 0 und $+1$. In der Praxis besteht kein Unterschied darin, ob die Sigmoidfunktion oder die tanh-Funktion genutzt wird. Noch eine andere Möglichkeit bildet die Gaußsche Funktion, welche die Euklidische Distanz statt des Skalarproduktes für die Ähnlichkeit nutzt; wir befassen uns mit solchen Netzen radialer Basisfunktionen in Kapitel 12.

Die Ausgabe ist eine lineare Kombination von Werten der nichtlinearen Basisfunktionen, die durch die verborgenen Einheiten berechnet werden. Es ist festzustellen, dass die verborgenen Einheiten eine nichtlineare Transformation vom d-dimensionalen Eingaberaum in den H-dimensionalen Raum vornehmen, der durch die verborgenen Einheiten aufgespannt wird, und in diesem Raum implementiert die zweite Ausgabeschicht eine lineare Funktion.

Es gibt keine Einschränkung auf nur eine verborgene Schicht. Weitere verborgene Schichten mit eigenen verborgenen Einheiten und Gewichten der Eingaben für selbige können sukzessive hinzugefügt werden. Wenn die Einheiten der ersten hinzugefügten verborgenen Schicht die Sigmoidfunktion verwenden, um ihre Ausgaben zu berechnen, berechnen diese nichtlineare, komplexere Funktionen ihrer Eingaben und damit der Eingaben des MLP. In der Praxis greift man nur selten auf mehr als eine verborgene Schicht zurück, da die Analyse eines Netzes mit vielen verborgenen Schichten sich als recht kompliziert erweist; wenn in manchen Fällen jedoch die verborgene Schicht zu viele verborgene Einheiten enthält, ist es ratsam, zu multiplen verborgenen Schichten überzugehen, wobei „lange und schmale“ Netze den „kurzen und dicken“ vorgezogen werden.

11.6 Das MLP als universelle Näherungsfunktion

Wir können jede beliebige Boolesche Funktion als eine Disjunktion von Konjunktionen darstellen, und solch ein Boolescher Ausdruck kann durch ein mehrlagiges Perzeptron mit einer verborgenen Schicht implementiert werden. Jede Konjunktion wird durch eine verborgene Einheit und die Disjunktion durch die Ausgabeeinheit implementiert. Ein Beispiel wäre:

$$x_1 \text{ XOR } x_2 = (x_1 \text{ AND } \sim x_2) \text{ OR } (\sim x_1 \text{ AND } x_2)\,.$$

Wir haben bereits gesehen, wie wir AND und OR mit Hilfe von Perzeptronen implementieren können. Zwei Perzeptronen können also parallel die zwei AND-Elemente implementieren, und ein weiteres Perzeptron obenauf kann die OR-Verbindung herstellen (siehe Abbildung 11.6). Wir sehen, dass die erste Schicht Eingaben aus dem (x_1, x_2)- in den (z_1, z_2)-Raum abbildet, der durch die Perzeptronen der ersten Schicht definiert ist. Man beachte, dass sowohl die Eingabe (0,0) als auch (1,1) auf (0,0) im (z_1, z_2)-Raum abgebildet werden, wodurch die lineare Trennbarkeit in diesem zweiten Raum ermöglicht wird.

Im binären Fall definieren wir somit für jede Eingabekombination mit Ausgabe 1 eine verborgene Einheit, die auf diese bestimmte Konjunktion der Eingabe hin prüft. Die Ausgabeschicht implementiert dann die Disjunktion. Es sei darauf hingewiesen, dass es sich hierbei nur um einen Existenznachweis handelt und dass solche Netze möglicherweise nicht praktisch verwertbar sind, da bei d Eingaben bis zu 2^d verborgene Einheiten nötig sein können. So eine Architektur implementiert die Methode der Lookup-Tabellen und generalisiert nicht.

UNIVERSELLE APPROXIMATION

STÜCKWEISE KONSTANTE APPROXIMATION

Wir können dies auf den Fall kontinuierlicher Eingaben erweitern, um zu zeigen, dass auf ähnliche Weise jede beliebige willkürliche Funktion mit kontinuierlicher Eingabe und Ausgaben mit einem mehrlagigen Perzeptron approximiert werden kann. Der Beweis für die *universelle Approximation* ist leicht bei zwei verborgenen Schichten: für jeden Eingabefall oder jede Region kann eben jene Region durch Hyperebenen auf allen Seiten abgegrenzt werden, indem verborgene Einheiten in der ersten verborgenen Schicht genutzt werden. Eine verborgene Einheit in der zweiten Schicht fügt diese dann mittels AND zusammen, um die Region abzugrenzen. Wir setzen dann das Gewicht für die Verbindung von jener verborgenen Einheit zur Ausgabeeinheit auf den gewünschten Funktionswert. Dies resultiert in einer *stückweisen konstanten Annäherung* der Funktion; das ist äquivalent zur Nichtbeachtung aller Terme in der Taylorschen Erweiterung außer dem konstanten Term. Die Genauigkeit kann auf den gewünschten Wert erhöht werden, indem die Anzahl an verborgenen Einheiten vergrößert wird, wodurch ein feineres Raster auf die Eingabe angewendet wird. Man beachte, dass keine formellen Grenzen bezüglich der Anzahl an benötigten

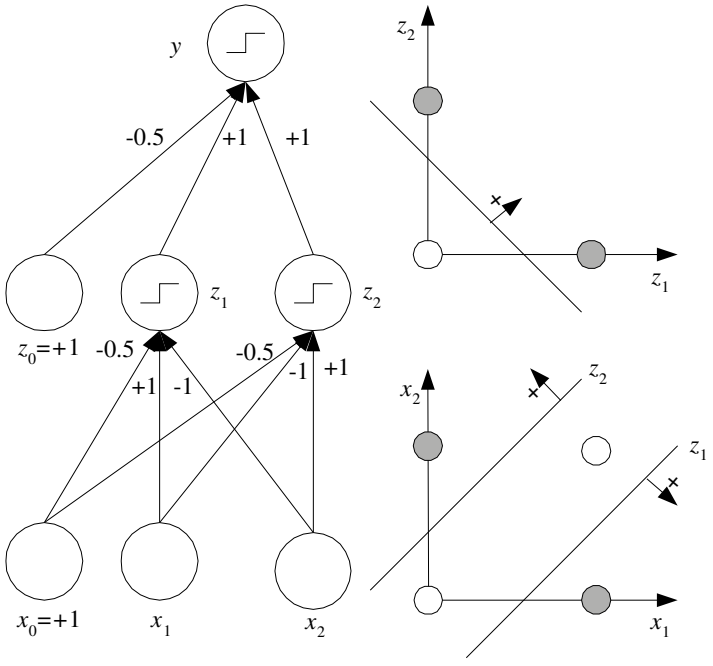


***Abb. 11.6*:** Das mehrlagige Perzeptron, welches das XOR-Problem löst. Die verborgenen Einheiten und die Ausgabe besitzen die Schwellwertaktivierungsfunktion mit dem Schwellwert 0.

verborgenen Einheiten vorgegeben sind. Diese Eigenschaft versichert uns nur erneut, dass eine Lösung existiert; ansonsten hilft uns dies nicht weiter. Es wurde bewiesen, dass ein MLP mit *einer* verborgenen Schicht (mit einer willkürlichen Anzahl an verborgenen Einheiten) jede beliebige nichtlineare Funktion der Eingabe lernen kann (Hornik, Stinchcombe, und White 1989).

11.7 Backpropagation-Algorithmus

Das Training eines mehrlagigen Perzeptrons entspricht dem Training eines einfachen Perzeptrons; der einzige Unterschied besteht darin, dass jetzt die Ausgabe dank der nichtlinearen Basisfunktion in den verborgenen Einheiten eine nichtlineare Funktion der Eingabe ist. Wenn die verborgenen Einheiten als Eingaben betrachtet werden, ist die zweite Schicht ein Perzeptron und wir wissen bereits, wie wir die Parameter v_{ij} in diesem Fall bei gegebenen Eingaben h_j aktualisieren können. Für

die Gewichte w_{hj} der ersten Schicht nutzen wir die Kettenregel, um den Gradienten zu berechnen:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}} .$$

Backpropagation

Es ist, als ob der Fehler von der Ausgabe y zurück zu den Eingaben propagiert wird, woraus die Bezeichung *Backpropagation* herstammt (Rumelhart, Hinton, und Williams 1986a).

11.7.1 Nichtlineare Regression

Betrachten wir als erstes den Fall der nichtlinearen Regression (mit einer einzelnen Ausgabe), die durch

$$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0 \tag{11.13}$$

berechnet wird, wobei z_h sich durch Gleichung 11.11 ergibt. Die Fehlerfunktion über die gesamte Stichprobe bei der Regression ist

$$E(\mathbf{W}, \boldsymbol{v}|\mathcal{X}) = \frac{1}{2}\sum_t (r^t - y^t)^2 . \tag{11.14}$$

Die zweite Schicht ist ein Perzeptron mit verborgenen Einheiten als Eingaben, und wir nutzen die Methode der kleinsten Quadrate, um die Gewichte der zweiten Schicht zu aktualisieren:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t . \tag{11.15}$$

Die erste Schicht entspricht ebenfalls Perzeptronen mit verborgenen Einheiten als Ausgabeeinheiten, jedoch können wir bei der Aktualisierung der Gewichte der ersten Schicht die Methode der kleinsten Quadrate nicht direkt verwenden, da uns keine für die verborgenen Einheiten spezifizierte gewünschte Ausgabe vorliegt. Hier kommt die Kettenregel ins Spiel. Wir schreiben

$$\begin{aligned}
\Delta w_{hj} &= -\eta \frac{\partial E}{\partial w_{hj}} \\
&= -\eta \sum_t \frac{\partial E^t}{\partial y^t}\frac{\partial y^t}{\partial z_h^t}\frac{\partial z_h^t}{\partial w_{hj}} \\
&= -\eta \sum_t \underbrace{-(r^t - y^t)}_{\partial E^t/\partial y^t} \underbrace{v_h}_{\partial y^t/\partial z_h^t} \underbrace{z_h^t(1 - z_h^t)x_j^t}_{\partial z_h^t/\partial w_{hj}} \\
&= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t .
\end{aligned} \tag{11.16}$$

Das Produkt der ersten beiden Terme $(r^t - y^t)v_h$ agiert wie der Fehlerterm für die verborgene Einheit h. $(r^t - y^t)$ ist der Fehler in der Ausgabe wird von dort zurückgreicht und gewichtet durch die „Verantwortlichkeit“ der verborgenen Einheit, welche durch deren Gewicht v_h gegeben ist. Im dritten Term ist $z_h(1 - z_h)$ die Ableitung der Sigmoidfunktion und x_j^t ist die Ableitung der gewichteten Summe hinsichtlich des Gewichts w_{hj}. Man beachte, dass die Veränderung des Gewichts der ersten Schicht, Δw_{hj}, das Gewicht der zweiten Schicht, v_h, verwendet. Somit sollten wir die Änderungen in beiden Schichten berechnen und die Gewichte der ersten Schicht aktualisieren und dabei den *alten* Wert der Gewichte der zweiten Ebene nutzen; erst dann sind die Gewichte der zweiten Schicht zu aktualisieren.

Die Gewichte w_{hj}, v_h werden anfangs auf kleine zufällige Werte gesetzt, beispielsweise im Intervall $[-0{,}01, 0{,}01]$, um die Sigmoidfunktionen nicht zu sättigen. Es ist auch eine gute Idee, die Eingaben zu normalisieren, so dass sie alle den Mittelwert 0 und Varianz 1 haben und die gleiche Skalierung besitzen, da wir einen einzelnen η-Parameter nutzen.

Mit den hier gegebenen Lerngleichungen berechnen wir für jedes Muster die Richtung, in welcher jeder Parameter verändert werden muss sowie den Umfang der Änderung. Beim *Batch-Lernen* sammeln wir diese Änderungen über alle Muster und bringen die Änderung einmal ein, sobald ein kompletter Durchlauf der gesamten Stichprobe stattgefunden hat, so wie dies bereits in den vorigen Aktualisierungsgleichungen gezeigt wurde. Einen kompletten Durchlauf durch alle Muster in der Trainingsmenge bezeichnet man als *Epoche*. Das online-Lernen ist ebenfalls möglich, indem die Gewichte nach jedem Muster aktualisiert werden, wodurch der stochastische Gradientenabstieg implementiert wird. Der Lernfaktor η sollte in diesem Fall kleiner gewählt werden und Muster sollten in zufälliger Reihenfolge eingelesen werden. Das online-Lernen führt zu schnellerer Konvergenz, da es ähnliche Muster im Datensatz geben kann. In diesem Fall hat die Stochastizität einen ähnlichen Effekt wie das Hinzufügen von Rauschen und kann dadurch helfen, lokalen Minima zu entgehen.

Batch-Lernen

Epoche

Ein Beispiel für das Training eines mehrlagigen Perzeptrons für die Regression ist in Abbildung 11.7 dargestellt. Mit fortschreitendem Training nähert sich die MLP-Anpassung der zugrunde liegenden Funktion an und der Fehler nimmt ab (siehe Abbildung 11.8). Abbildung 11.9 zeigt, wie die MLP-Anpassung als Summe der Ausgaben der verborgenen Einheiten geformt wird.

Es ist auch möglich, mehrere Ausgabeeinheiten zu verwenden; in diesem Fall werden eine gewisse Anzahl an Regressionsproblemen zur gleichen Zeit gelernt. Wir erhalten

$$y_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \tag{11.17}$$

und der Fehler ist

$$E(\mathbf{W},\mathbf{V}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \,. \tag{11.18}$$

Die Regeln zur Batch-Aktualisierung sind somit

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t \,, \tag{11.19}$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih}\right] z_h^t (1 - z_h^t) x_j^t \,. \tag{11.20}$$

$\sum_i (r_i^t - y_i^t) v_{ih}$ entspricht dem akkumulierten rückpropagierten Fehler der verborgenen Einheit h aus allen Ausgabeeinheiten. Der Pseudocode ist im Listing 11.2 gegeben. Man beachte, dass in diesem Fall alle Ausgabeeinheiten dieselben verborgenen Einheiten gemein haben und somit dieselbe verborgene Darstellung nutzen. Eine Alternative besteht darin, separate mehrlagige Perzeptronen für die separaten Regressionsprobleme zu trainieren, jedes dabei mit seinen eigenen separaten verborgenen Einheiten.

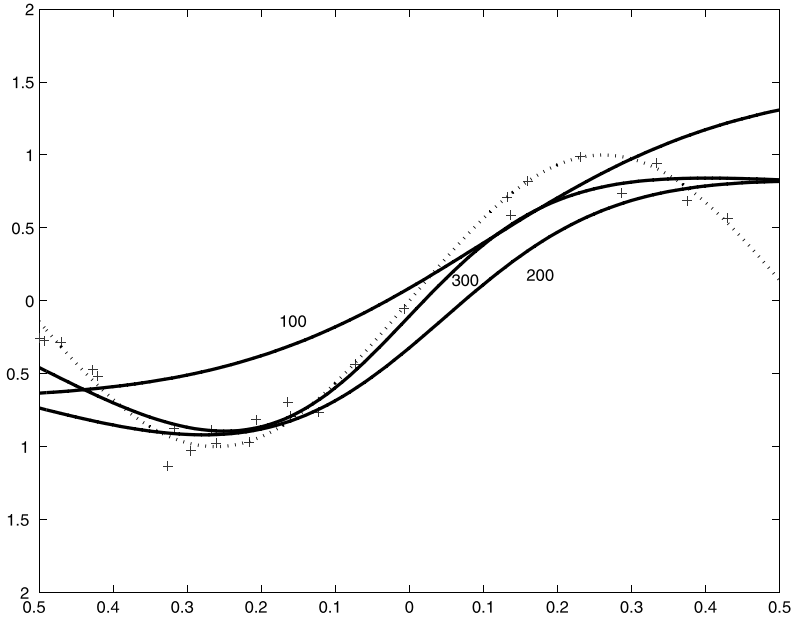


Abb. 11.7: Mit „+“ dargestellt sind die Stichprobentrainingsdaten, wobei $x^t \sim U(-0{,}5, 0{,}5)$ und $y^t = f(x^t) + \mathcal{N}(0, 0{,}1)$. $f(x) = \sin(6x)$ ist durch eine gestrichelte Linie dargestellt. Die Entwicklung der Anpassung eines MLP mit zwei verborgenen Einheiten nach 100, 200 und 300 Epochen ist eingezeichnet.

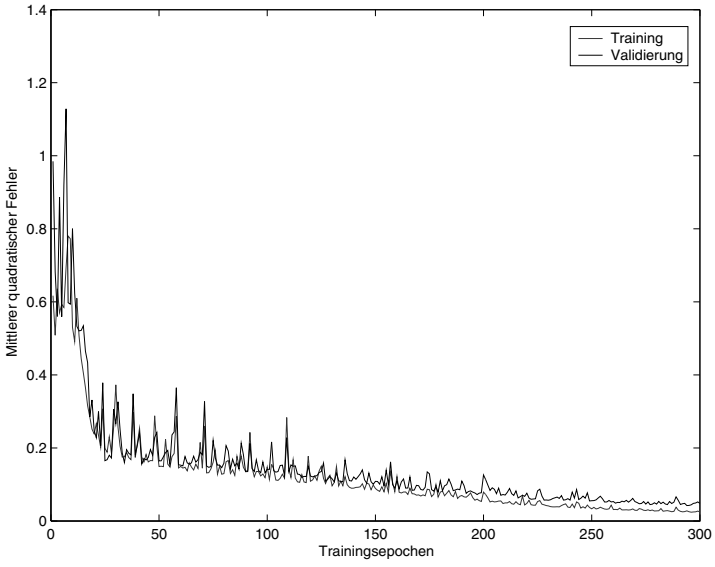


***Abb. 11.8*:** Der mittlere quadratische Fehler am Trainings- und Validierungsdatensatz als eine Funktion von Trainingsepochen.

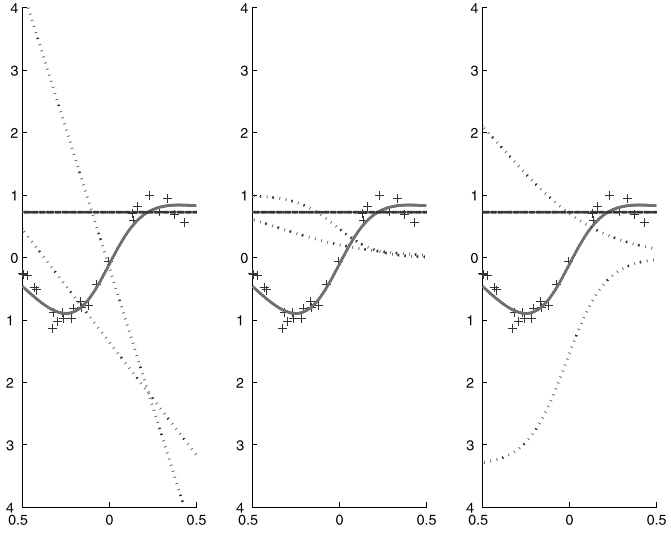


***Abb. 11.9*:** (a) Die Hyperebenen definiert durch die Gewichte der verborgenen Einheiten der ersten Schicht, (b) Ausgaben der verborgenen Einheiten und (c) Ausgaben der verborgenen Einheiten multipliziert mit den Gewichten der zweiten Schicht. Zwei leicht verschobene verborgene Sigmoideinheiten, die eine davon multipliziert mit einem negativen Gewicht, implementieren eine Schwelle, wenn sie hinzugefügt werden. Mit mehr verborgenen Einheiten kann eine bessere Approximation erzielt werden (siehe Abbildung 11.10).

11.7.2 Zweiklassendiskriminanz

Wenn es zwei Klassen gibt, so reicht eine Ausgabeeinheit aus:

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0\right) , \tag{11.21}$$

wodurch $P(\mathcal{C}_1|\boldsymbol{x}^t)$ und $\hat{P}(\mathcal{C}_2|\boldsymbol{x}^t) \equiv 1 - y^t$ approximiert werden. Wir erinnern uns aus Abschnitt 10.7, dass die Fehlerfunktion in diesem Fall wie folgt aussieht:

$$E(\mathbf{W}, \boldsymbol{v}|\mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) . \tag{11.22}$$

Die Aktualisierungsgleichungen entsprechend des Gradientenabstiegs lauten

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t , \tag{11.23}$$

$$\Delta w_{hj} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t . \tag{11.24}$$

```
Initialisiere alle v_ih und w_hj mit rand(−0,01, 0,01)
Repeat
    For alle (x^t, r^t) ∈ X in zufälliger Reihenfolge
        For h = 1, ..., H
            z_h ← sigmoid(w_h^T x^t)
        For i = 1, ..., K
            y_i = v_i^T z
        For i = 1, ..., K
            Δv_i = η(r_i^t − y_i^t) z
        For h = 1, ..., H
            Δw_h = η(Σ_i (r_i^t − y_i^t) v_ih) z_h (1 − z_h) x^t
        For i = 1, ..., K
            v_i ← v_i + Δv_i
        For h = 1, ..., H
            w_h ← w_h + Δw_h
Until Konvergenz
```

***Listing 11.2*:** Backpropagation-Algorithmus für das Training eines mehrlagigen Perzeptrons für die Regression mit K Ausgaben. Dieser Code kann leicht für die Klassifikation bei zwei Klassen (durch Festlegung einer einzelnen sigmoidalen Ausgabe) und für die Klassifikation bei $K > 2$ Klassen (durch Nutzung von Softmax-Ausgaben) angepasst werden.

Wie beim einfachen Perzeptron sind die Aktualisierungsgleichungen für die Regression und Klassifikation identisch (was nicht bedeutet, dass es die Werte auch sind).

11.7.3 Diskriminanz bei multiplen Klassen

Bei einem Klassifikationsproblem mit $K > 2$ Klassen gibt es K Ausgaben

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \tag{11.25}$$

und wir nutzen Softmax, um die Abhängigkeit zwischen den Klassen anzudeuten, also dass sie sich gegenseitig ausschließen und vollständig sind:

$$y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} , \tag{11.26}$$

wobei $P(\mathcal{C}_i|\boldsymbol{x}^t)$ durch y_i approximiert wird. Die Fehlerfunktion lautet

$$E(\mathbf{W}, \mathbf{V}|\mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t \tag{11.27}$$

und wir erhalten die Aktualisierungsgleichungen unter Verwendung des Gradientenabstiegs:

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t , \tag{11.28}$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t . \tag{11.29}$$

Richard und Lippmann (1991) zeigten, dass für ein gegebenes Netz von ausreichender Komplexität und mit genügend Daten ein angemessen trainiertes mehrlagiges Perzeptron die a-posteriori-Wahrscheinlichkeiten schätzt.

11.7.4 Multiple verborgene Schichten

Wie wir bereits gesehen haben, ist es möglich mehrere verdeckte Schichten zu verwenden. In diesem Fall besitzt jede verdeckte Schicht ihre eigenen versteckten Einheiten und diese wiederum jeweils ihre eigenen Gewichte ihrer Eingaben. Jede dieser versteckten Einheiten berechnet aus der gewichteten Summe ihrer Eingaben mit Hilfe der Sigmoidfunktion ihre Ausgabe. Wenn wir einmal davon ausgehen, dass wir für die Regression

ein mehrlagiges Perzeptron verwenden mit zwei verborgenen Schichten, dann ergibt sich

$$z_{1h} = \text{sigmoid}(\boldsymbol{w}_{1h}^T \boldsymbol{x}) = \text{sigmoid}\Big(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\Big),\ h = 1, \ldots, H_1 ,$$

$$z_{2l} = \text{sigmoid}(\boldsymbol{w}_{2l}^T \boldsymbol{z}_1) = \text{sigmoid}\Big(\sum_{h=0}^{H_1} w_{2lh} z_{1h} + w_{2l0}\Big),\ l = 1, \ldots, H_2 ,$$

$$y = \boldsymbol{v}^T \boldsymbol{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0 .$$

Dabei entsprechen $\boldsymbol{w}_{1h}$ und $\boldsymbol{w}_{2l}$ den Gewichten der ersten bzw. zweiten Schicht, z_{1h} und z_{2h} sind die Einheiten auf der ersten und zweiten verborgenen Schicht, und $\boldsymbol{v}$ steht für die Gewichte der dritten Schicht. Das Training eines solchen Netzes ist ähnlich, außer dass wir eine weitere Schicht rückpropagieren müssen, um die Gewichte der ersten Schicht zu trainieren (Übung 5).

11.8 Trainingsprozeduren

11.8.1 Verbesserung der Konvergenz

Der Gradientenabstieg bietet diverse Vorteile. Er ist einfach. Er ist lokal, sprich, die Veränderung in einem Gewicht nutzt lediglich die Werte der präsynaptischen und postsynaptischen Einheiten und den Fehler (angemessen rückpropagiert). Beim online-Training muss der Datensatz nicht gespeichert werden und das Training kann sich an Veränderungen der zu lernenden Aufgabenstellung anpassen. Aus diesen Gründen kann (und wird) der Gradientenabstieg in der Hardware implementiert werden. Der Gradientenabstieg konvergiert jedoch nur langsam. Wenn beim Lernen die Zeit eine wichtige Rolle spielt, können ausgefeiltere Optimierungsmethoden verwendet werden (Battiti 1992). Bishop (1995) diskutiert die Anwendung von konjugierten Gradienten (conjugate gradient) und Methoden zweiter Ordnung beim Training mehrlagiger Perzeptronen im Detail. Allerdings existieren zwei oft verwendete einfache Techniken, welche die Leistung des Gradientenabstiegs deutlich verbessern und gradientenbasierte Methoden für reale Applikationen wirklich nutzbar machen.

Momentum. Nehmen wir an, w_i sei ein beliebiges Gewicht in einem mehrlagigen Perzeptron in einer beliebigen Schicht, inklusive der Verzerrungseingaben. Bei jeder Parameteraktualisierung sind sukzessive Werte für Δw_i^t unter Umständen derart unterschiedlich, dass große Oszillationen auftreten können und die Konvergenz nur langsam voranschreitet. t

ist der Zeitindex, welcher der Epochennummer beim Batch-Lernen und der Iterationsnummer beim online-Lernen entspricht. Der Gedanke ist der, einen gleitenden Durchschnitt zu bilden, indem die vorhergehende Aktualisierung in die aktuelle Veränderung eingearbeitet wird, als ob es ein *Momentum* aufgrund vorhergehender Aktualisierungen gibt: MOMENTUM

$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1} \,. \tag{11.30}$$

α wird allgemein zwischen 0,5 und 1,0 gewählt. Dieser Ansatz erweist sich als besonders nützlich beim online-Lernen, bei dem wir als Ergebnis einen durchschnittsbildenden Effekt erhalten und die Kurve während der Konvergenz glätten. Der Nachteil liegt darin, dass die vergangenen Werte für Δw_i^{t-1} zusätzlich gespeichert werden müssen.

Adaptive Lernrate. Beim Gradientenabstieg bestimmt der Lernfaktor η die Größe der Änderung der Parameter. Er wird allgemein zwischen 0,0 und 1,0 gewählt und ist zumeist kleiner oder gleich 0,2. Er kann zu Zwecken schnellerer Konvergenz adaptiv verwendet werden, so dass er anfänglich große Werte annimmt, wenn der Lernprozess voranschreitet und bei einer Verlangsamung des Lernprozesses ebenfalls verringert wird:

$$\Delta\eta = \begin{cases} +a & \text{falls } E^{t+\tau} < E^t \,, \\ -b\eta & \text{andernfalls} \,. \end{cases} \tag{11.31}$$

Somit erhöhen wir η um einen konstanten Betrag, wenn der Fehler am Trainingsdatensatz abnimmt und verringern ihn geometrisch, wenn er zunimmt. Da E von einer Epoche zur nächsten oszillieren kann, ist es ratsamer, den Durchschnitt der letzten vergangenen Epochen als E^t zu nutzen.

11.8.2 Übertraining

Ein mehrlagiges Perzeptron mit d Eingaben, H verborgenen Einheiten und K Ausgaben hat $H(d+1)$ Gewichte in der ersten Schicht und $K(H+1)$ Gewichte in der zweiten. Sowohl die Speicher- als auch die Rechenkomplexität eines MLP ist $\mathcal{O}(H \cdot (K+d))$. Sei e die Anzahl an Trainingsepochen, so ergibt sich die Rechenzeitkomplexität für das Training als $\mathcal{O}(e \cdot H \cdot (K+d))$.

In einer Anwendung sind d und K vordefiniert und H ist der Parameter, mit dem wir experimentieren können, um die Komplexität des Modells abzustimmen. Wir wissen aus vorhergehenden Kapiteln, dass sich ein zu komplexes Modell das Rauschen in einem Datensatz einprägt und nicht auf den Validierungsdatensatz hin generalisiert. Beispielsweise haben wir dieses Phänomen bereits im Falle der Polynomregression gesehen, wobei wir bemerkt haben, dass beim Vorhandensein von Rauschen oder

kleinen Stichproben die Erhöhung des Polynomgrades zu schlechterer Generalisierung führt. Ähnlich dazu verschlechtert sich bei einem MLP die Genauigkeit bei einer großen Anzahl an verborgenen Schichten (siehe Abbildung 11.10), und das Dilemma von Verzerrung und Varianz trifft genau wie auf jeden anderen statistischen Schätzer auch auf das MLP zu (Geman, Bienenstock, und Doursat 1992).

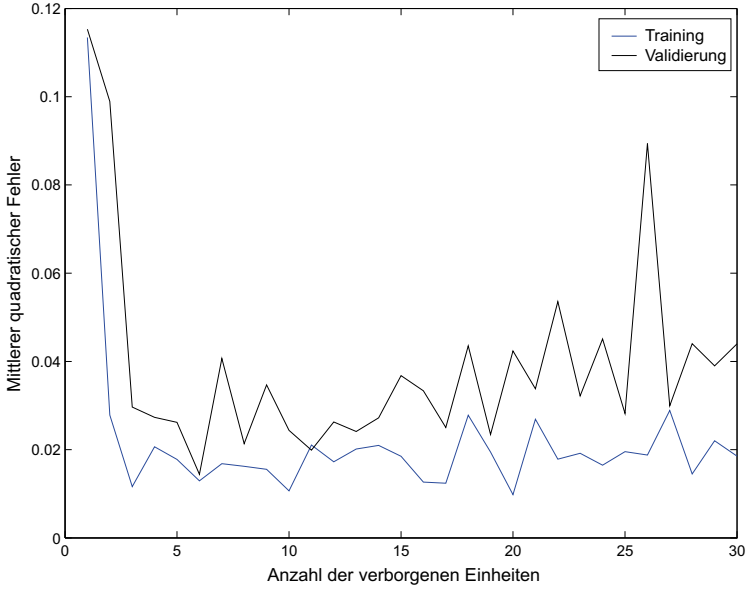


***Abb. 11.10*:** Mit zunehmender Komplexität bleibt der Trainingsfehler auf einem festen Wert, doch der Validierungsfehler beginnt zu wachsen und das Netz weist Anzeichen einer Überanpassung auf.

Vergleichbares Verhalten tritt auf, wenn das Training zu lang fortgesetzt wird: je mehr Trainingsepochen durchschritten werden, desto mehr verringert sich der Fehler am Trainingsdatensatz, jedoch erhöht sich der Fehler am Validierungsdatensatz ab einem bestimmten Punkt (siehe Abbildung 11.11). Es sei daran erinnert, dass zu Beginn alle Gewichte nahe 0 liegen und somit kaum einen Effekt besitzen. Mit voranschreitendem Training beginnen die wichtigsten Gewichte, sich von 0 wegzubewegen und werden verwertet. Wird das Training jedoch weiter fortgesetzt, um den Fehler am Trainingsdatensatz mehr und mehr zu verringern, dann werden beinahe alle Gewichte aktualisiert und entfernen sich von 0, wodurch sie an Einfluss gewinnen und damit zu effektiven Parametern werden. Es ist also, als ob mit fortschreitendem Training neue Parameter zum System hinzugefügt werden, was zu erhöhter Komplexität und schlechte-

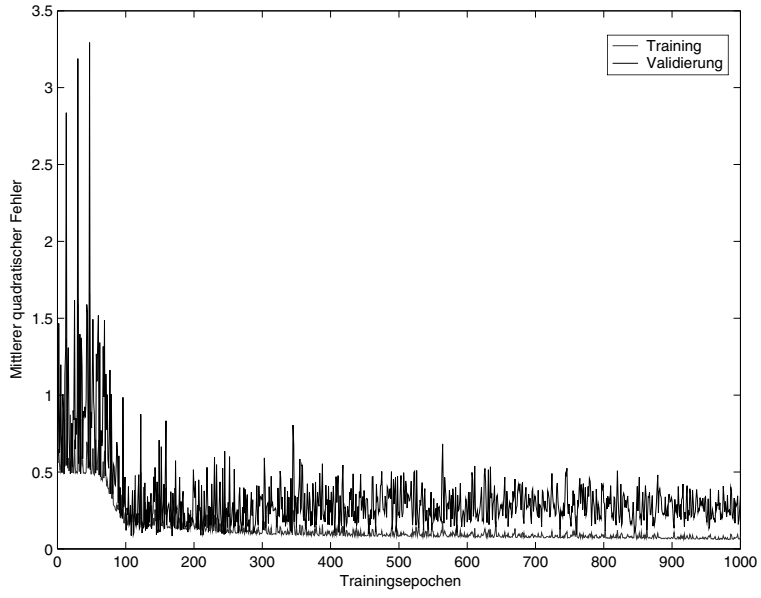


***Abb. 11.11*:** Mit fortschreitendem Training erhöht sich der Validierungsfehler und das Netz wird zu stark angepasst.

rer Generalisierung führt. Der Lernprozess sollte durch *frühes Stoppen* abgebrochen werden, um das Problem des *Übertrainings* abzufangen. Der optimale Zeitpunkt, um das Training zu beenden, sowie die optimale Anzahl an verborgenen Einheiten wird durch Kreuzvalidierung bestimmt, welche Tests der Leistungskraft des Netzes an im Training nicht verwendeten Validierungsdaten beinhaltet.

FRÜHES STOPPEN
ÜBERTRAINING

Wegen der Nichtlinearität hat die Fehlerfunktion viele Minima und der Gradientenabstieg konvergiert zum nächstgelegenen Minimum. Um den erwarteten Fehler einschätzen zu können, wird dasselbe Netz mehrfach trainiert, wobei mit unterschiedlichen Ausgangswerten begonnen wird und der Durchschnitt des Validierungsfehlers berechnet wird.

11.8.3 Strukturieren des Netzes

Bei einigen Anwendungen glauben wir möglicherweise, dass die Eingabe eine lokale Struktur aufweist. Bei visuellen Anwendungen beispielsweise wissen wir, dass naheliegende Pixel miteinander korrelieren und dass es lokale Merkmale wie Ränder und Ecken gibt. Jedes beliebige Objekt, zum Beispiel eine handgeschriebene Ziffer, kann als eine Kombination solcher primitiven Elemente gesehen werden. Ähnlich dazu ist bei der Sprache Lokalität in der Zeit zu finden und zeitlich nah beieinander liegende Eingaben können als Sprachprimitive gruppiert werden. Durch

die Kombination dieser primitiven Elemente können längere Äußerungen, zum Beispiel Sprachphoneme, definiert werden. In so einem Fall werden beim Entwurf des MLP die verborgenen Einheiten nicht mit allen Eingabeeinheiten verbunden, da nicht alle Eingaben korrelieren. Stattdessen definieren wir verborgene Einheiten, welche ein Fenster über dem Eingaberaum definieren und nur mit dieser kleinen lokalen Teilmenge der Eingaben verbunden sind. Dies verringert die Zahl an Verbindungen und somit auch die Zahl der freien Paramter (Le Cun et al. 1989).

Wir können dies in aufeinanderfolgenden Schichten wiederholen, wobei jede Schicht mit einer kleinen Zahl darunterliegender verborgener Einheiten verbunden ist und auf ein komplizierteres Merkmal testet, indem sie die Merkmale ihrer Vorgänger in einem größeren Teil des Eingaberaums kombiniert, bis wir die Ausgabeschicht erreichen (siehe Abbildung 11.12). Beispielsweise könnten die Eingaben als Pixel vorliegen. Indem die Pixel betrachtet werden, könnten die ersten Einheiten verborgener Schichten lernen, auf Kanten diverser Ausrichtungen zu testen. Indem dann die Kanten kombiniert werden, können die Einheiten der zweiten verborgenen Schicht lernen, nach Kombinationen von Kanten zu suchen – zum Beispiel Bögen, Ecken, Anfangs- und Endpunkte von Geraden – und diese dann durch die Kombination in höheren Schichten nutzen, um nach Halbkreisen, Rechtecken, oder im Fall der Gesichtserkennung nach Augen, Mund usw. zu suchen. Hierbei handelt es sich um ein Beispiel für einen *hierarchischen Kegel*, bei dem Merkmale umso komplexer, abstrakter und weniger in ihrer Anzahl werden, je weiter man in den Schichten von der Eingabeschicht kommend aufsteigt bis letztlich zur Ausgabeschicht, die den Merkmalen dann Klassen zuordnet. Eine solche Struktur wird als *faltendes neuronales Netz* bezeichnet, wobei die Arbeit jeder einzelnen verborgenen Einheit als Faltung ihrer Eingabe mit ihrem Gewichtsvektor aufgefasst wird. Eine ältere, ähnliche Struktur ist das *Neocognitron* (Fukushima 1980).

HIERARCHISCHER KEGEL

FALTENDES NEURONALES NETZ

NEOCOGNITRON

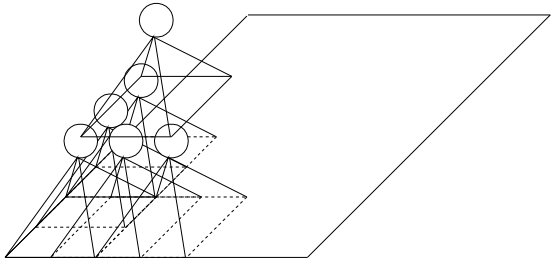


***Abb. 11.12*:** Ein strukturiertes MLP. Jede Einheit ist mit einer lokalen Gruppe von unter ihr liegenden Einheiten verbunden und testet bei visuellen Anwendungen auf ein bestimmtes Merkmal – zum Beispiel Ränder, Ecken und so weiter. Nur eine verborgene Einheit ist für jede Region zu sehen. Typischerweise existieren viele davon, um auf verschiedene lokale Merkmale zu testen.

In so einem Fall können wir die Anzahl an Parametern durch *gemeinsame Gewichte* weiter reduzieren. Wenn wir noch einmal das Beispiel der visuellen Erkennung von Elementen heranziehen, sehen wir, dass sich bei der Suche nach Merkmalen, wie beispielsweise ausgerichteten Rändern, selbige an mehreren Stellen im Eingaberaum befinden können. Statt also unabhängige verborgene Einheiten zu definieren, welche verschiedene Merkmale an verschiedenen Stellen im Eingaberaum lernen, können wir Kopien derselben verborgenen Einheiten nutzen, die verschiedene Teile des Eingaberaums durchsuchen (siehe Abbildung 11.13). Während des Lernprozesses berechnen wir die Gradienten, indem wir verschiedene Eingaben verwenden, bilden dann den Durchschnitt dieser und bringen eine einzelne Aktualisierung ein. Dies impliziert die Existenz eines einzelnen Parameters, welcher die Gewichte an multiplen Verbindungen definiert. Da die Aktualisierung an einem Gewicht außerdem auf Gradienten für mehrere Eingaben basiert, ist es, als ob der Trainingsdatensatz effektiv multipliziert wird.

GEMEINSAME GEWICHTE

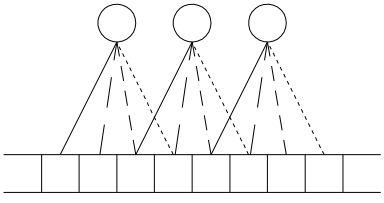


Abb. 11.13: Bei gemeinsamen Gewichten haben verschiedene Einheiten Verbindungen zu unterschiedlichen Eingaben, teilen sich jedoch denselben Gewichtswert (dargestellt durch den Linientyp). Nur eine Menge an Einheiten ist abgebildet; tatsächlich existieren mehrere Mengen von Einheiten, wobei jede nach anderen Merkmalen sucht.

11.8.4 Hinweise

Die Kenntnis der lokalen Struktur erlaubt es uns, das mehrlagige Netz vorzustrukturieren, und durch gemeinsame Gewichte wird die Zahl der Parameter verringert. Die Alternative eines MLP mit vollständig verbundenen Schichten hat keine solche Struktur und ist schwieriger zu trainieren. Jegliches in Beziehung zur Anwendung stehendes Wissen sollte wann immer möglich in die Netzstruktur eingebracht werden. Hierbei spricht man von *Hinweisen* (Abu-Mostafa 1995) und sie stellen Eigenschaften der Zielfunktion dar, welche uns unabhängig von den Trainingsbeispielen bekannt sind.

HINWEISE

Bei der Bilderkennung existieren Invarianzhinweise: die Identität eines Objektes ändert sich nicht, wenn man es rotiert, verschiebt oder skaliert (siehe Abbildung 11.14). Hinweise sind Hilfsinformationen, die genutzt werden können, um den Lernprozess zu lenken und die besonders nützlich sind, wenn der Trainingsdatensatz begrenzt ist. Es gibt verschiedene Arten, wie Hinweise genutzt werden können:

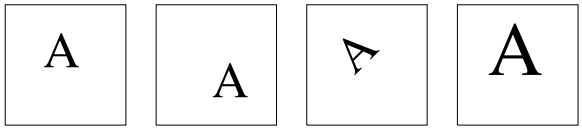


Abb. 11.14: Die Identität des Objektes ändert sich nicht, wenn man es verschiebt, rotiert oder skaliert. Man beachte, dass dies nicht unbedingt immer gilt oder nur bis zu einem gewissen Grad zutrifft: „b“ und „q“ sind rotierte Versionen voneinander. Dies sind Hinweise, die in den Lernprozess eingearbeitet werden können, um das Lernen zu vereinfachen.

VIRTUELLE BEISPIELE

1. Hinweise können genutzt werden, um *virtuelle Beispiele* zu erstellen. Mit dem Wissen, dass das Objekt gegenüber Skalierungen invariant ist, können wir beispielsweise anhand eines Trainingsbeispiels verschiedene Kopien mit unterschiedlichen Skalierungen erstellen und sie zum Trainingsdatensatz mit dem gleichen Label hinzufügen. Hier besteht der Vorteil darin, dass wir den Trainingsdatensatz vergrößern, ohne den Lerner in irgendeiner Form modifizieren zu müssen. Probleme können sich daraus ergeben, dass unter Umständen zu viele Beispiele nötig sein werden, um dem Lerner die Invarianz beizubringen.

2. Die Invarianz kann als eine Vorbearbeitungsstufe implementiert werden. Zum Beispiel besitzen Lesegeräte für optische Schriftzeichen eine Vorbearbeitungsstufe, in der das Bild des Eingabezeichens zentriert und hinsichtlich Größe und Strichführung normalisiert wird. Sofern möglich, stellt dies die einfachste Lösung dar.

3. Der Hinweis kann in die Netzstruktur eingebracht werden. Lokale Strukturierung und gemeinsame Gewichte, was wir in Abschnitt 11.8.3 behandelt haben, sind ein Beispiel, durch das wir Invarianz hinsichtlich kleiner Verschiebungen und Rotationen erzielen können.

4. Der Hinweis kann auch eingearbeitet werden, indem die Fehlerfunktion modifiziert wird. Nehmen wir an, wir wissen, dass $\boldsymbol{x}$ und $\boldsymbol{x}'$ aus Perspektive der Anwendung identisch sind, wobei $\boldsymbol{x}'$ ein „virtuelles Beispiel“ von $\boldsymbol{x}$ sein kann. Das heißt, $f(\boldsymbol{x}) = f(\boldsymbol{x}')$, wenn $f(\boldsymbol{x})$ die von uns zu approximierende Funktion darstellt. Sei $g(\boldsymbol{x}|\theta)$ die Bezeichnung für unsere Näherungsfunktion, beispielsweise ein MLP mit θ als Gewichte. Dann definieren wir für alle Paare der Form $(\boldsymbol{x}, \boldsymbol{x}')$ die Straffunktion

   $$E_h = [g(\boldsymbol{x}|\theta) - g(\boldsymbol{x}'|\theta)]^2$$

   und fügen sie als zusätzlichen Term zur gewöhnlichen Fehlerfunktion hinzu:

   $$E' = E + \lambda_h \cdot E_h \,.$$

Hierbei handelt es sich um einen Strafterm, der all die Fälle bestraft, bei denen unsere Vorhersagen dem Hinweis nicht folgen; λ_h ist das Gewicht solch einer Strafe (Abu-Mostafa 1995).

Ein weiteres Beispiel ist der Approximationshinweis. Gehen wir davon aus, dass wir den exakten Wert $f(x)$ für x nicht kennen, jedoch wissen, dass er im Intervall $[a_x, b_x]$ liegt. Somit ist unser hinzugefügter Strafterm gleich

$$E_h = \begin{cases} 0 & \text{falls } g(x|\theta) \in [a_x, b_x] \,, \\ (g(x) - a_x)^2 & \text{falls } g(x|\theta) < a_x \,, \\ (g(x) - b_x)^2 & \text{falls } g(x|\theta) > b_x \,. \end{cases}$$

Dies ähnelt der Fehlerfunktion, die bei der Support-Vektor-Regression (Abschnitt 13.10) verwendet wurde und die geringe Näherungsfehler toleriert.

Und noch ein weiteres Beispiel findet sich in der *Tangent-Prop-Methode* (Simard et al. 1992), bei der die Transformation bezüglich der wir den Hinweis definieren – zum Beispiel eine Drehung um einen bestimmten Winkel –, durch eine Funktion modelliert wird. Die übliche Fehlerfunktion wird modifiziert (indem ein weiterer Term hinzugefügt wird), so dass sich Parameter ändern können ohne den Fehler zu verändern, wenn die dadurch implizierten Änderungen laut Invarianz keine Änderung des Objektes bedeuten.

TANGENT-PROP-METHODE

11.9 Anpassung der Netzgröße

Wir haben gesehen, dass zu große Netze mit zu vielen freien Parametern oft nur unzureichend vom Trainingsdatensatz generalisieren. Um die optimale Netzgröße zu bestimmen, besteht die am häufigsten verwendete Methode darin, viele verschiedene Architekturen auszuprobieren, sie alle am Trainingsdatensatz zu trainieren und diejenige auszuwählen, welche am besten für den Validierungsdatensatz generalisiert. Ein anderer Ansatz besteht darin, diese *strukturelle Adaption* in den Lernalgorithmus einzuarbeiten. Dafür gibt es zwei Möglichkeiten:

STRUKTURELLE ADAPTION

1. Beim *destruktiven* Ansatz beginnen wir mit einem großen Netz und entfernen schrittweise überflüssige Einheiten und/oder Verbindungen.
2. Beim *konstruktiven* Ansatz beginnen wir mit einem kleinen Netz und fügen schrittweise Einheiten und/oder Verbindungen hinzu, um die Leistungsfähigkeit zu steigern.

Eine destruktive Methode ist der *Gewichtsabbau*, bei dem die Idee darin besteht, überflüssige Verbindungen zu entfernen. Damit wir in der Lage

GEWICHTSABBAU

sind, zu bestimmen, ob eine Einheit oder Verbindung notwendig ist oder nicht, müssen wir im Idealfall einmal mit und einmal ohne die jeweilige Einheit bzw. Verbindung trainieren und den Unterschied im Fehler anhand eines separaten Validierungsdatensatzes überprüfen. Das ist kostspielig, da es für alle Kombinationen solcher Einheiten/Verbindungen durchgeführt werden sollte.

Vorausgesetzt, dass eine Verbindung nicht benutzt wird, wenn ihr Gewicht null ist, versehen wir jedes Gewicht mit der Tendenz zu null zu streben, so dass diese verschwindet falls sie nicht explizit *erhöht* wird, *um den Fehler zu minimieren*. Für jedes Gewicht w_i im Netz nutzen wir die Aktualisierungsregel

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i \,. \tag{11.32}$$

Das ist äquivalent zur Anwendung des Gradientenabstiegs auf die Fehlerfunktion mit einem hinzugefügten Strafterm, der Netze mit vielen Gewichten ungleich 0 bestraft:

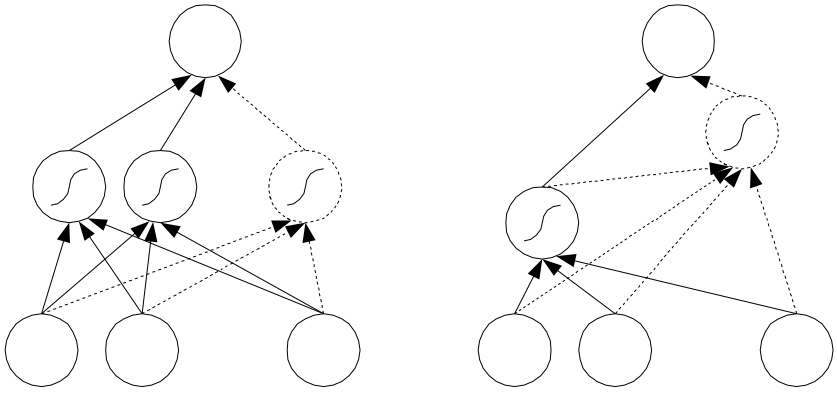
$$E' = E + \frac{\lambda}{2} \sum_i w_i^2 \,. \tag{11.33}$$

Einfachere Netze generalisieren besser. Das ist ein Hinweis, den wir durch Hinzufügen eines Strafterms implementieren. Es sei darauf hingewiesen, dass dies nicht heißt, dass einfache Netze immer besser sind als große Netze; es bedeutet lediglich, dass bei Vorliegen zweier Netze mit dem gleichen Trainingsfehler das einfachere – nämlich das mit weniger Gewichten – eine höhere Wahrscheinlichkeit besitzt, für den Validierungsdatensatz besser zu generalisieren.

Der Effekt des zweiten Terms in Gleichung 11.32 ähnelt dem einer Sprungfeder, welche jedes Gewicht auf 0 zieht. Beginnend mit einem Wert nahe 0 – es sei denn, der tatsächliche Fehlergradient ist groß und verursacht eine Aktualisierung – fällt das Gewicht aufgrund des zweiten Terms allmählich ganz in Richtung 0 ab. λ ist der Parameter, welcher die relative Wichtigkeit des Fehlers am Trainingsdatensatz und die Komplexität durch Parameter ungleich 0 repräsentiert; somit bestimmt er die Geschwindigkeit des Abfallens: für große λ werden die Gewichte auf 0 heruntergezogen, egal welchen Wert der Trainingsfehler annimmt; bei kleinen λ gibt es kaum Bestrafung für Gewichte ungleich 0. λ wird mit Hilfe der Kreuzvalidierung feinabgestimmt.

Statt mit einem großen Netz zu beginnen und überflüssige Verbindungen oder Einheiten zu *kürzen*, kann man auch an einem kleinen Netz ansetzen und Einheiten und zugehörige Verbindungen hinzufügen, wann immer der Bedarf dafür besteht (Abbildung 11.15). Bei der *dynamischen Knotengenerierung* (Ash 1989) wird ein MLP mit einer verborgenen Schicht mit einer verborgenen Einheit trainiert, und falls der Fehler nach dem

DYNAMISCHE KNOTENGENERIERUNG



Dynamische Knotengenerierung *Kaskadierende Korrelation*

***Abb. 11.15*:** Zwei Beispiele für konstruktive Algorithmen. Bei der dynamischen Knotengenerierung wird eine Einheit zu einer existierenden Schicht hinzugefügt. Die kaskadierende Korrelation addiert jede Einheit als neue verborgene Schicht, die mit allen vorausgehenden Schichten verbunden ist. Gestrichelte Linien stehen für die neu hinzugefügten Einheiten/Verbindungen. Verzerrungseinheiten/-gewichtungen sind aus Gründen der Einfachheit nicht dargestellt.

Konvergieren immer noch groß ist, wird eine weitere verborgene Einheit hinzugefügt. Die eingehenden Gewichte der neu hinzugefügten Einheit und ihr ausgehendes Gewicht werden zufällig initialisiert und mit den vorher existierenden Gewichten trainiert, die nicht erneut initialisiert werden und an ihren vorhergehenden Werten anknüpfen.

KASKADIERENDE KORRELATION

Bei der *kaskadierenden Korrelation* (Fahlman und Lebiere 1990) ist jede hinzugefügte Einheit eine neue verborgene Einheit in einer weiteren verborgenen Schicht. Jede verborgene Schicht hat nur eine Einheit, die mit allen ihr vorausgehenden Einheiten und den Eingaben verbunden ist. Die vorher existierenden Gewichte werden eingefroren und nicht trainiert; lediglich die eingehenden und ausgehenden Gewichte der neu hinzugefügten Einheit werden trainiert.

Bei der dynamischen Knotengenerierung wird eine neue verborgene Einheit zu einer existierenden Schicht hinzugefügt, aber niemals eine weitere verborgene Schicht. Bei der kaskadierenden Korrelation wird immer eine neue verborgene Schicht mit einer einzelnen Einheit addiert. Die ideale konstruktive Methode sollte in der Lage sein, zu entscheiden, wann eine neue verborgene Schicht zu erstellen und wann eine Einheit einer existierenden Schicht hinzuzufügen ist. Das stellt noch immer ein offenes Forschungsproblem dar.

Inkrementelle Algorithmen sind ebenfalls interessant, da sie eine Modifikation sowohl der Parameter als auch der Modellstruktur während des Lernprozesses umfassen. Denken wir zum Beispiel an einen Raum, der

durch die Struktur eines mehrlagigen Perzeptrons definiert ist, und an Operatoren für das Hinzufügen/Entfernen von Einheiten oder Schichten, mit denen wir uns in diesem Raum bewegen (Aran et al. 2009). Inkrementelle Algorithmen führen dann eine Suche in diesem Zustandsraum aus, indem die Operatoren (gemäß einer bestimmten Regel) angewendet werden und das Ergebnis in Abhängigkeit von einem Gütemaß (beispielsweise eine Kombination aus Komplexität und Validierungsfehler) akzeptiert oder abgelehnt wird. Ein anderes Beispiel haben wir bei der polynomialen Regression, wenn Terme höherer Ordnung automatisch während der Trainingsphase hinzugefügt/entfernt werden, um die Modellkomplexität an die Datenkomplexität anzupassen. Mit sinkenden Kosten für den Rechenaufwand sollte eine derartige automatische Modellselektion ein Teil des Lernprozesses werden, welcher automatisch und ohne Eingreifen des Nutzers stattfindet.

11.10 Bayessche Betrachtungsweise des Lernens

Der Bayessche Ansatz für das Training neuronaler Netze betrachtet die Parameter, sprich die Verbindungsgewichtungen w_i, als zufällige Variablen, die aus einer a-priori-Verteilung $p(w_i)$ entnommen wurden, und berechnet die a-posteriori-Wahrscheinlichkeit anhand der Daten:

$$p(\boldsymbol{w}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{X})} \,, \tag{11.34}$$

wobei $\boldsymbol{w}$ der Vektor aller Gewichte des Netzes ist. Die MAP-Schätzung $\hat{\boldsymbol{w}}$ ist der Modalwert der a-posteriori-Wahrscheinlichkeit

$$\hat{\boldsymbol{w}}_{MAP} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \log p(\boldsymbol{w}|\mathcal{X}) \,. \tag{11.35}$$

Logarithmiert man Gleichung 11.34, so erhalten wir

$$\log p(\boldsymbol{w}|\mathcal{X}) = \log p(\mathcal{X}|\boldsymbol{w}) + \log p(\boldsymbol{w}) + C \,.$$

Der erste Term auf der rechten Seite ist die Log-Likelihood und der zweite Term entspricht dem Logarithmus der a-priori-Wahrscheinlichkeit. Sind die Gewichte unabhängig und wird die a-priori-Wahrscheinlichkeit als Gaußverteilung angenommen, $\mathcal{N}(0{,}1/2\lambda)$, so ist

$$p(\boldsymbol{w}) = \prod_i p(w_i) \text{ mit } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right] \,. \tag{11.36}$$

Die MAP-Schätzung minimiert die erweiterte Fehlerfunktion

$$E' = E + \lambda \|\boldsymbol{w}\|^2 \,, \tag{11.37}$$

wobei E dem üblichen Klassifikations- oder Regressionsfehler entspricht (negative Log-Likelihood). Dieser erweiterte Fehler ist exakt die Fehlerfunktion, die wir beim Gewichtsabbau genutzt haben (Gleichung 11.33). Die Verwendung eines großen λ setzt geringe Variabilität in den Parametern voraus, wendet eine stärkere Kraft auf sie an, um sie auf einen Wert nahe 0 herunterzudrücken, und beachtet die a-priori-Wahrscheinlichkeit stärker als die Daten; ist λ klein, so vergrößert sich die erlaubte Variabilität der Parameter. Dieser Ansatz zur Entfernung unnötiger Parameter ist in der Statistik auch als *Ridge-Regression* bekannt.

RIDGE-REGRESSION

Es handelt sich hierbei um ein weiteres Beispiel für die *Regularisierung* mit einer Kostenfunktion, bei der die Anpassung an Daten und die Modellkomplexität kombiniert werden:

REGULARISIERUNG

$$\text{Kosten} = \text{Datenfehlanpassung} + \lambda \cdot \text{Komplexität} \,. \tag{11.38}$$

Die Verwendung der Bayesschen Schätzung beim Training mehrlagiger Perzeptronen wird bei MacKay 1992a, b behandelt. Die Bayessche Schätzung werden wir in Kapitel 16 ausführlicher behandeln.

Aus empirischer Sicht wurde festgestellt, dass nach dem Training die meisten Gewichte eines mehrlagigen Perzeptrons normal um 0 herum verteilt liegen, was die Anwendung der Methode des Gewichtsabbaus rechtfertigt. Doch das muss nicht immer der Fall sein. Nowlan und Hinton (1992) entwickelten die Methode des *Soft Weight Sharing* , bei der Gewichte aus einer Mischung aus Gaußverteilungen gezogen werden, wodurch es ihnen möglich wird, multiple Cluster zu bilden. Außerdem könnten diese Cluster an beliebiger Stelle – nicht nur um 0 herum – zentriert werden und modifizierbare Varianzen besitzen. Dies verändert die a-priori-Wahrscheinlichkeit aus Gleichung 11.36 hin zu einer Mischung von $M \geq 2$ gaußverteilten Wahrscheinlichkeiten

SOFT WEIGHT SHARING

$$p(w_i) = \sum_{j=1}^{M} \alpha_j p_j(w_i) \,, \tag{11.39}$$

wobei α_j den a-priori-Wahrscheinlichkeiten entspricht und $p_j(w_i) \sim \mathcal{N}(m_j, s_j^2)$ die Gaußkomponenten sind. M wird durch den Nutzer festgelegt und α_j, m_j, s_j werden anhand der Daten gelernt. Erweitert man die Fehlerfunktion um einen solchen logaritmierten Prior, konvergieren die Gewichte zum einen, um den Fehler zu verringern, und zum anderen, um den logarithmierten Prior zu erhöhen, indem sie gleichzeitig gruppiert (zu Cluster zusammengefügt) werden.

11.11 Dimensionalitätsreduktion

Wenn bei einem mehrlagigen Perzeptron die Anzahl an verborgenen Einheiten kleiner ist als die Anzahl an Eingaben, vollführt die erste

Schicht eine Dimensionalitätsreduktion. Die Form dieser Reduktion und des neu durch die verborgenen Einheiten aufgespannten Raums hängen davon ab, wofür das MLP trainiert wurde. Handelt es sich um ein MLP für die Klassifikation, bei dem die Ausgabeeinheiten der verborgenen Schicht folgen, so wird der neue Raum definiert und die Abbildung gelernt, um den Klassifikationsfehler zu minimieren (siehe Abbildung 11.16).

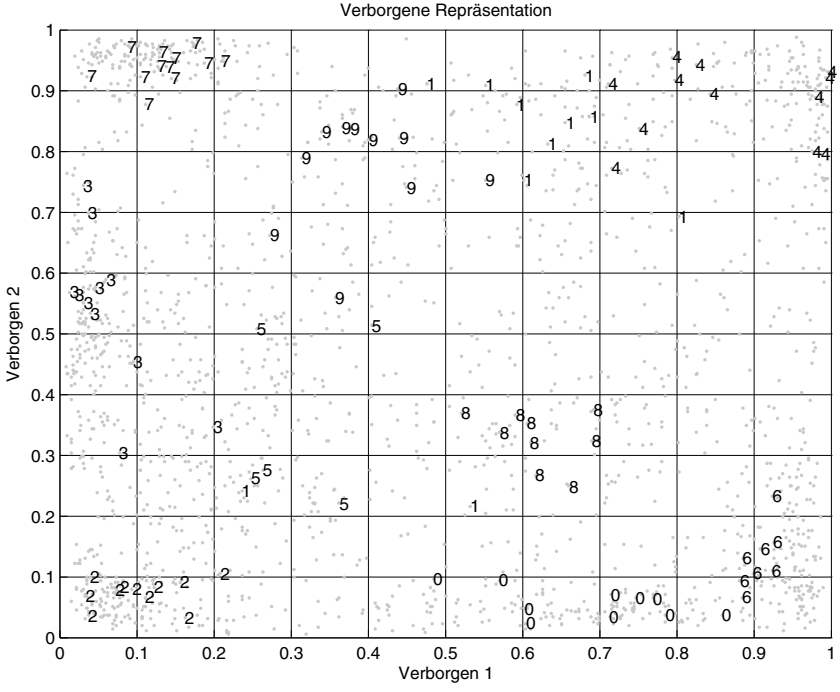


***Abb. 11.16*:** Dargestellt sind Optdigits-Daten, die in den Raum der zwei verborgenen Einheiten eines für die Klassifikation trainierten MLP transformiert wurden. Nur die Label für 100 Datenpunkte sind dargestellt. Dieses MLP mit 64 Eingaben, zwei verborgenen Einheiten und 10 Ausgaben besitzt 80-prozentige Genauigkeit. Aufgrund der Sigmoidfunktion liegen die Werte der verborgenen Einheiten zwischen 0 und 1, und die Klassen sind um die Ecken herum gruppiert. Dieses Diagramm kann mit denen aus Kapitel 6 verglichen werden, welche unter Verwendung anderer Methoden zur Dimensionalitätsreduktion mit demselben Datensatz erstellt wurden.

Wir können eine Vorstellung von dem, was das MLP tut, erlangen, indem wir die Gewichte analysieren. Wir wissen, dass das Skalarprodukt maximal ist, wenn die zwei Vektoren identisch sind. Somit können wir uns vorstellen, dass jede verborgene Einheit ein Muster durch ihre eingehenden Gewichte definiert, und indem wir diese Muster analysieren, können wir Wissen aus einem trainierten MLP extrahieren. Sind die Eingaben normalisiert, liefern uns die Gewichte Aussagen zu deren relativer Wichtigkeit. Derlei

Analyse ist nicht einfach, vermittelt uns jedoch ein gewisses Verständnis dessen, was ein MLP tut und erlaubt uns somit einen Blick in die „Black Box“.

Eine interessante Architektur bildet der *Autoencoder* (Cottrell, Munro und Zipser 1987). Dabei handelt es sich um eine MLP-Architektur, bei der es so viele Ausgaben gibt wie Eingaben und die benötigten Ausgaben gleich den Eingaben definiert werden (Abbildung 11.17). Um die Eingaben in der Ausgabeschicht reproduzieren zu können, wird das MLP gezwungen, die beste Repräsentation der Eingaben in der verborgenen Schicht zu finden. Ist die Anzahl der verborgenen Einheiten geringer als die der Eingaben, führt dies zu einer Dimensionalitätsreduktion. Sobald das Training vollendet wurde, agiert die erste Schicht von der Eingabe zur verborgenen Schicht als Codierer und die Werte der verborgenen Einheiten bilden die codierte Repräsentation. Die zweite Schicht von den verborgenen Einheiten zu den Ausgabeeinheiten agiert als Decodierer und rekonstruiert das Originalsignal aus seiner codierten Darstellung.

AUTOENCODER

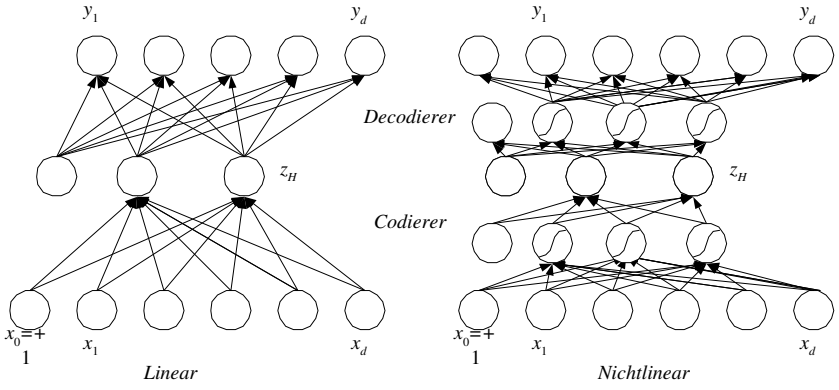


***Abb. 11.17*:** Im sogenannten Autoencoder gibt es genauso viele Ausgabe- wie Eingabeeinheiten und die gewünschten Ausgaben entsprechen den Eingaben. Ist die Anzahl an verborgenen Einheiten geringer als die Zahl an Eingaben, wird das MLP trainiert, um die beste Codierung der Eingaben an den verborgenen Einheiten zu finden; damit wird eine Dimensionalitätsreduktion vollzogen. Im linken Bild agiert die erste Schicht als Codierer und die zweite Schicht als Decodierer. Wenn wie im rechten Bild Codierer und Decodierer mehrlagige Perzeptronen mit sigmoidalen verborgenen Einheiten sind, so vollführt das Netz eine nichtlineare Dimensionalitätsreduktion.

Es ist bewiesen (Bourlard und Kamp 1988), dass ein Autoencoder-MLP mit einer verborgenen Schicht von Einheiten die Hauptkomponentenanalyse implementiert (Abschnitt 6.3). Allerdings entsprechen die Gewichte der verborgenen Einheiten nicht Eigenvektoren, die mit Hilfe ihrer Eigenwerte nach ihrer Wichtigkeit geordnet sind, sondern sie spannen den gleichen Raum auf, wie die H hauptsächlichen Eigenvektoren. Sind Codierer und Decodierer keine einlagigen, sondern mehrlagige Perzeptronen mit sigmoi-

daler Nichtlinearität in den verborgenen Einheiten, so implementiert der Codierer eine nichtlineare Dimensionalitätsreduktion. In Abschnitt 11.13 betrachten wir „tiefe Netze“, die aus mehreren nichtlinearen verborgenen Schichten zusammengesetzt sind.

Eine weitere Möglichkeit, ein MLP zur Dimensionalitätsreduktion zu verwenden, ist die multidimensionale Skalierung (siehe Abschnitt 6.7). Mao und Jain (1995) veranschaulichen, wie ein MLP verwendet werden kann, um das *Sammon-Mapping* zu lernen. Rufen wir uns Gleichung 6.37 ins Gedächtnis zurück, so definiert sich der Sammon-Stress als

SAMMON-MAPPING

$$E(\theta|\mathcal{X}) = \sum_{r,s} \left[\frac{\|\boldsymbol{g}(\boldsymbol{x}^r|\theta) - \boldsymbol{g}(\boldsymbol{x}^s|\theta)\| - \|\boldsymbol{x}^r - \boldsymbol{x}^s\|}{\|\boldsymbol{x}^r - \boldsymbol{x}^s\|} \right]^2 . \tag{11.40}$$

Ein MLP mit d Eingaben, H verborgenen Einheiten und $k < d$ Ausgabeeinheiten wird verwendet, um $g(\boldsymbol{x}|\theta)$ zu implementieren, wobei die d-dimensionale Eingabe auf einen k-dimensionalen Vektor abgebildet wird und θ den Gewichten des MLP entspricht. Bei einem Datensatz von $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ können wir den Sammon-Stress mittels Gradientenabstieg direkt minimieren, um das MLP zu lernen, also $g(\boldsymbol{x}|\theta)$, so dass die Entfernungen zwischen den k-dimensionalen Darstellungen der Originalpunkte so nah wie möglich die Entfernungen im Originalraum widerspiegeln.

11.12 Lernen mit Zeitreihen

Bis jetzt haben wir uns nur mit Fällen beschäftigt, bei denen die Eingaben mit einem Mal alle zusammen eingespeist werden. Bei einigen Anwendungen ist die Eingabe zeitlich variabel, so dass wir eine temporale Sequenz lernen müssen. Bei anderen Szenarien kann sich auch die Ausgabe mit der Zeit ändern. Beispiele hierfür sind:

- *Sequenzerkennung.* Hierbei handelt es sich um die Zuweisung einer gegebenen Sequenz zu einer von mehreren Klassen. Die Spracherkennung ist ein Beispiel, bei dem die Eingabesignalsequenz der gesprochenen Sprache entspricht und die Ausgabe sich in der Codierung der gesprochenen Wörter findet. Das heißt, die Eingabe ändert sich in der Zeit, die Ausgabe tut dies jedoch nicht.

- *Sequenzreproduktion.* Nachdem ein Teil einer gegebenen Sequenz vorgelegt wurde, sollte hier das System den Rest vorhersagen. Zeitreihenvorhersagen sind ein Beispiel hierfür, bei denen die Eingabe gegeben ist, sich aber die Ausgabe verändert.

- *Temporale Assoziation.* Dies ist der allgemeinste Fall, bei dem eine bestimmte Ausgabesequenz als Ausgabe nach einer spezifischen

Eingabesequenz gegeben wird. Die Eingabe- und Ausgabesequenzen können verschieden sein. Hier ändern sich sowohl Eingabe als auch Ausgabe in der Zeit.

11.12.1 Time Delay Neural Networks

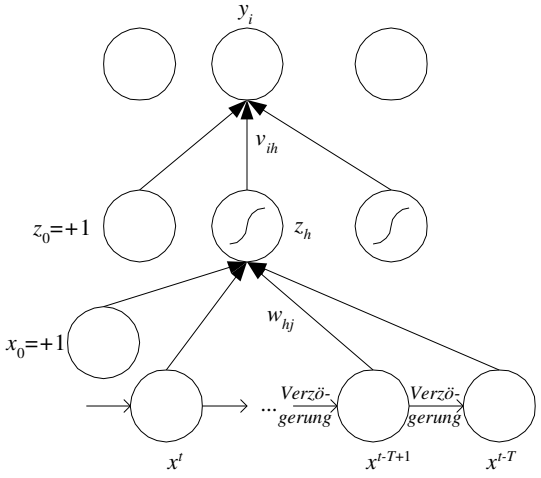


***Abb. 11.18*:** Ein Time Delay Neural Network (TDNN). Eingaben innerhalb eines Zeitfensters von Länge T werden zeitlich verzögert, bis wir alle T Eingaben als Eingabevektor in das MLP einspeisen können.

Der einfachste Weg, eine temporale Sequenz zu erkennen, besteht darin, sie in eine räumliche Sequenz zu übertragen. Dann kann jede bisher diskutierte Methode für die Klassifikation verwendet werden. Bei einem *Time Delay Neural Network* (TDNN) (Waibel et al. 1989), also einem *neuronalen Netz mit Zeitverzögerungspuffer*, werden vorangegangene Eingaben zeitlich verzögert, um sie mit den finalen Eingaben zu synchronisieren, woraufhin alles zusammen als Eingabe in das System eingespeist wird (siehe Abbildung 11.18). Rückpropagierung kann dann eingesetzt werden, um die Gewichte zu trainieren. Um zeitlich lokale Merkmale zu extrahieren, kann man Schichten von strukturierten Verbindungen und gemeinsame Gewichte nutzen, um Invarianz gegenüber zeitlichen Verschiebungen zu erreichen. Die hauptsächliche Einschränkung dieser Architektur besteht darin, dass die Größe des Zeitfensters, welches wir über die Sequenz schieben, a priori festgelegt werden sollte.

TIME DELAY NEURAL NETWORK

11.12.2 Rekurrente Netze

Bei einem *rekurrenten Netz* haben die Einheiten zusätzlich zu ihren Vorwärtsverbindungen außerdem Verbindungen zu sich selbst oder Verbindungen mit den Einheiten in der vorangegangen Schicht. Diese Rekur-

REKURRENTES NETZ

renz agiert als Kurzzeitgedächtnis und ermöglicht es dem Netz, sich an vergangene Ereignisse zu erinnern.

Am häufigsten verwendet man ein teilweise rekurrentes Netz, bei dem eine begrenzte Anzahl an rekurrenten Verbindungen einem mehrlagigen Perzeptron hinzugefügt werden (siehe Abbildung 11.19). Dies kombiniert den Vorteil der Fähigkeit zur nichtlinearen Approximation eines MLP mit der Fähigkeit zur temporalen Repräsentation durch die Rekurrenz. Solch ein Netz kann verwendet werden, um jede der drei temporalen Assoziationsaufgaben (s.o.) zu erfüllen. Es ist auch möglich, verborgene Einheiten in den rekurrenten Rückwärtsverbindungen zu nutzen, welche dann als *Kontexteinheiten* bezeichnet werden. Es sind keine formalen Ergebnisse bekannt, die genutzt werden könnten, um die beste Architektur für eine bestimmte gegebene Applikation zu bestimmen.

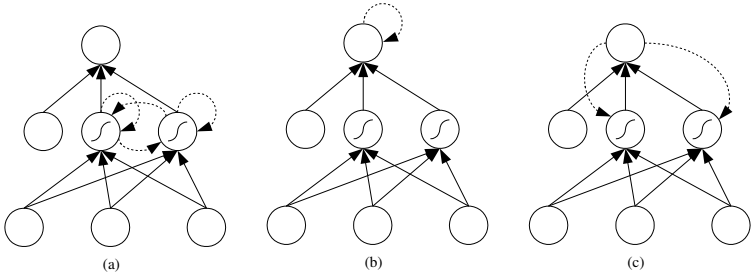


***Abb. 11.19*:** Beispiel eines MLP mit partieller Rekurrenz. Rekurrente Verbindungen werden durch gestrichelte Linien dargestellt: (a) Eigenverbindungen in der verborgenen Schicht, (b) Eigenverbindungen in der Ausgabeschicht und (c) Verbindungen von der Ausgabe zur verborgenen Schicht. Kombinationen hiervon sind auch möglich.

ZEITLICHE ENTFALTUNG

Besitzen die Sequenzen eine geringe maximale Länge, dann kann die *zeitliche Entfaltung* genutzt werden, um ein willkürliches rekurrentes Netz in ein äquivalentes Feedforward-Netz zu konvertieren (siehe Abbildung 11.20). Eine separate Einheit und Verbindung wird für Kopien zu unterschiedlichen Zeitpunkten erstellt. Das resultierende Netz kann mittels Rückpropagierung mit der Zusatzbedingung trainiert werden, dass Kopien einer jeden Verbindung identisch erhalten bleiben sollten. Die Lösung besteht wie bei den gemeinsamen Gewichten darin, die verschiedenen Gewichtsveränderungen im Laufe der Zeit zu summieren und das Gewicht um den Durchschnitt zu verändern. Dies bezeichnet man als *backpropagation through time*, also als *Rückpropagierung durch die Zeit* (Rumelhart, Hinton und Willams 1986b). Das Problem bei diesem Ansatz ergibt sich aus der Speicheranforderung bei langen Sequenzen. Der Algorithmus des *rekurrenten Echtzeit-Lernens* (engl. *real time recurrent learning*) (Williams und Zipser 1989) wird für das Training von rekurrenten Netzen ohne zeitliche Entfaltung verwendet und hat den Vorteil, dass er für Sequenzen willkürlicher Länge eingesetzt werden kann.

RÜCKPROPAGIERUNG DURCH DIE ZEIT

REKURRENTES ECHTZEIT-LERNEN

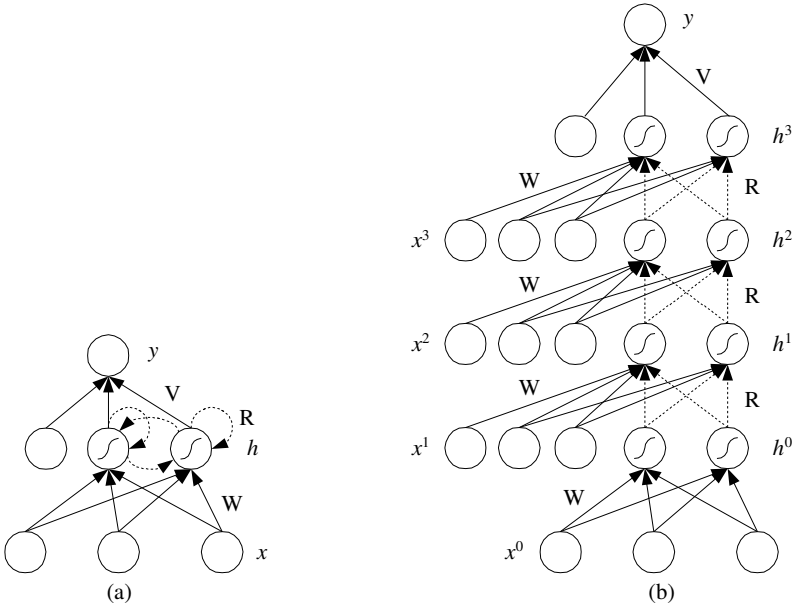


***Abb. 11.20*:** Backpropagation durch die Zeit: (a) rekurrentes Netz und (b) sein äquivalentes entfaltetes Netz, das sich in vier Schritten identisch verhält.

11.13 Tiefes Lernen

Wenn ein lineares Modell nicht genügt, dann besteht eine Möglichkeit darin, neue Merkmale zu definieren, die nichtlineare Funktionen der Eingabe sind (zum Beispiel Funktionen, die Terme höherer Ordnung enthalten) und dann ein lineares Modell in dem Raum dieser Merkmale zu konstruieren (siehe Abschnitt 10.2). Dazu müssen wir wissen, welche Funktionen im konkreten Fall geeignete Basisfunktionen sind. Eine andere Möglichkeit ist die Verwendung eines Verfahrens der Merkmalsextraktion (beispielsweise PCA oder Isomap, siehe Kapitel 6), um den neuen Raum zu lernen. Diese Verfahren haben den Vorteil, dass sie auf Daten trainiert wurden. Der beste Ansatz scheint dennoch die Verwendung eines MLP zu sein, das solche Merkmale in seinen verborgenen Schichten extrahiert. Das MLP hat den Vorteil, dass die erste Schicht (Merkmalsextraktion) und die zweite Schicht (Kombination dieser Merkmale, um die Ausgabe vorherzusagen) zusammen in einer gekoppelten und überwachten Art und Weise gelernt werden.

Ein MLP mit einer verborgenen Schicht hat eine beschränkte Kapazität, während ein MLP mit mehreren verborgenen Schichten auch kompliziertere Funktionen der Eingabe lernen kann. Das ist die Idee hinter *tiefen neuronalen Netzen* (engl. *deep neural networks*), in denen, ausgehend von der Roheingabe, jede verborgene Schicht die Werte ihrer Vorgängerschicht

TIEFE NEURONALE NETZE

kombiniert und auf diese Weise kompliziertere Funktionen der Eingabe lernt.

Ein weiterer Aspekt tiefer Netze besteht darin, dass aufeinanderfolgende Schichten zunehmend abstrakteren Repräsentationen entsprechen, d. h., in der Ausgabeschicht repräsentieren die gelernten Ausgaben die abstraktesten Konzepte.

Ein Beispiel hierfür haben wir im Zusammenhang mit faltenden neuronalen Netzen (Abschnitt 11.8.3) gesehen, wo wir, ausgehend von Pixeln zunächst Kanten, dann Ecken und schließlich eine Ziffer bekommen haben. Es ist jedoch Anwenderwissen erforderlich, um die Konnektivität und die Gesamtarchitektur zu definieren. Betrachten wir ein MLP zur Gesichtserkennung, bei dem die Eingaben Bildpixel sind und jede verborgene Einheit mit allen anderen Eingaben verbunden ist. In diesem Fall hat das Netz kein Wissen darüber, dass es sich bei den Eingaben um Bilder von Gesichtern handelt, ja, noch nicht einmal, dass die Eingabe zweidimensional ist – die Eingabe ist einfach nur ein Vektor mit irgendwelchen Werten. Der Einsatz eines faltenden Netzes, in dem in die verborgenen Einheiten lokalisierte zweidimensionale Abschnitte eingespeist werden, ist eine Möglichkeit, diese Information so zu verwerten, dass korrekte Abstraktionen gelernt werden.

TIEFES LERNEN

Beim *tiefen Lernen* (engl. *deep learning*) besteht die Idee darin, Merkmale von zunehmendem Abstraktionsniveau unter minimaler Beteiligung des Menschen zu lernen (Bengio 2009). Der Hintergrund ist, dass wir bei den meisten Anwendungen nicht wissen, welche Struktur in der Eingabe steckt, und dass alle Formen von Abhängigkeiten während des Trainings automatisch aufgedeckt werden sollten. Es ist diese Extraktion von Abhängigkeiten, Mustern oder Regelmäßigkeiten, die das Abstrahieren und Lernen von allgemeinen Deskriptoren gestattet.

Ein Hauptprobleme beim Trainieren eines MLP mit mehreren verborgenen Schichten besteht darin, dass wir bei der Backpropagation des Fehlers in eine frühere Schicht die Ableitungen in allen Schichten anschließend multiplizieren müssen und der Gradient verschwindet. Das ist auch der Grund, weshalb entfaltete rekurrente Netze (Abschnitt 11.12.2) sehr langsam lernen. Bei faltenden Netzen ist dies nicht der Fall, da die Ein- und Ausgabefächer von verborgenen Einheiten typischerweise klein sind.

In einem tiefen neuronalen Netz wird gewöhnlich jeweils eine Schicht trainiert (Hinton und Salakhutdinov 2006). Das Ziel jeder einzelnen Schicht ist es, die hervorstechenden Merkmale der Daten zu extrahieren, mit denen sie gefüttert wird. Zu diesem Zweck kann ein Verfahren wie der in Abschnitt 11.11 diskutierte Autoencoder eingesetzt werden, was den zusätzlichen Vorteil hat, dass wir hierfür Daten ohne Zuordnung verwenden können. Ausgehend von der Roheingabe trainieren wir also einen Autoencoder, und die in seiner verborgenen Schicht gelernte Repräsentation wird dann als Eingabe zum Trainieren des nächsten Autoencoders benutzt usw., bis wir schließlich die finale Schicht erreichen, die mit zugeordneten

Daten überwacht trainiert wird. Nachdem alle Schichten eine nach der anderen auf diese Weise trainiert wurden, werden sie alle zusammengefügt und das gesamte Netz wird mit zugeordneten Daten feinabgestimmt.

Wenn eine große Menge von Daten und viel Rechenleistung zur Verfügung steht, kann das gesamte tiefe Netz überwacht trainiert werden, doch es besteht Einigkeit darüber, dass die Verwendung eines unüberwachten Verfahrens zum Initialisieren der Gewichte viel besser funktioniert als eine zufällige Initialisierung – das Lernen kann viel schneller und mit weniger zugeordneten Daten erfolgen.

Methoden des tiefen Lernens sind vor allem deshalb attraktiv, weil sie mit weniger manuellen Eingriffen auskommen. Es ist nicht nötig, die richtigen Merkmale oder geeignete Basisfunktionen (oder Kernel, siehe Kapitel 13) zu basteln, und wir müssen uns auch keine Gedanken um die richtigen Netzwerkarchitektur machen. Wenn wir einmal die Daten haben (und heutzutage haben wir „Big“ Data) und ausreichend viel Rechenkapazität zur Verfügung steht, dann können wir einfach warten und den Lernalgorithmus alles Nötige selbst aufdecken lassen.

Die dem tiefen Lernen zugrundeliegende Idee, mehrere Schichten von zunehmendem Abstraktionsniveau zu verwenden, ist naheliegend. Nicht nur beim maschinellen Sehen – zum Beispiel beim Erkennen handgeschriebener Ziffern oder bei der Gesichtserkennung –, sondern auch bei vielen anderen Anwendungen sind Abstraktionsschichten vortstellbar, und es wäre interessant, solche abstrakten Repräsentationen zu finden, unter anderem, weil dies eine Visualisierung und auch eine bessere Beschreibung des Problems gestatten würde.

Betrachten wir das maschinelle Übersetzen. Nehmen wir zum Beispiel an, wir beginnen mit einem englischen Satz. In mehreren Schichten der Verarbeitung und Abstraktion, die automatisch aus einem sehr großen Korpus englischer Sätze gelernt werden, um die lexikalischen, syntaktischen und semantischen Regeln der englischen Sprache zu codieren, würden wir für diesen Satz die abstrakteste Repräsentation erhalten. Betrachten wir nun den gleichen Satz auf Französisch. Die Verarbeitungsschichten, die aus einem französischen Textkorpus gelernt wurden, wären sehr verschieden von den englischen, doch wenn zwei Sätze auf der abstraktesten, sprachunabhängigen Ebene das gleiche bedeuten, dann sollten sie sehr ähnliche Repräsentationen haben.

11.14 Anmerkungen

Die Forschung in Richtung künstlicher neuronaler Netze ist so alt wie der digitale Computer. McCulloch und Pitts (1943) beschrieben das erste mathematische Modell für das künstliche Neuron. Rosenblatt (1962) legte 1962 das Perzeptronenmodell und einen Lernalgorithmus vor. Minsky und Papert (1969) zeigten die Einschränkungen von einlagigen Perzeptronen

auf, beispielsweise das XOR-Problem; und da es damals keinen Algorithmus zum Training eines mehrlagigen Perzeptrons mit einer verborgenen Schicht gab, wurden die Forschungsarbeiten bis auf wenige Stellen fast vollkommen eingestellt. Die Wiedergeburt der neuronalen Netze kam mit der Arbeit von Hopfield (1982). Gefolgt wurde diese von dem zweibändigen Buch zum Parallel Distributed Processing (PDP), verfasst durch die PDP-Forschungsgruppe (Rumelhart und McClelland 1986). Es scheint, als ob die Methode der Rückpropagierung unabhängig an mehreren Stellen beinahe zur gleichen Zeit erfunden wurde und die Einschränkungen eines einlagigen Perzeptrons nicht mehr länger zutrafen.

Beginnend Mitte der 1980er Jahre gab es eine explosionsartige Schaffensphase zu den Modellen künstlicher neuronaler Netze in diversen Disziplinen: Physik, Statistik, Psychologie, Kognitionswissenschaft, Neurowissenschaft und Linguistik, ganz zu schweigen von der Informatik, dem Elektroingenieurswesen und dem Gebiet der adaptiven Steuerung. Vielleicht ist der wichtigste Forschungsbeitrag zu künstlichen neuronalen Netzen die Synergie, mit welcher die diversen Disziplinen ineinander übergingen, speziell die Statistik in das Ingenieurswesen. Ihr ist es zu verdanken, dass das Gebiet des maschinellen Lernens heute weitgehend etabliert ist.

Das Feld ist inzwischen sehr gereift; Ziele sind moderater und besser definiert. Einer der Kritikpunkte an der Rückpropagierung bestand darin, dass sie biologisch nicht plausibel sei! Zwar ist der Begriff des „neuronalen Netzes“ immer noch weit verbreitet, jedoch ist man sich im Allgemeinen darüber einig, dass Modelle neuronaler Netze, beispielsweise mehrlagige Perzeptronen, nichtparametrische Schätzer sind, die am besten durch statistische Methoden analysiert werden.

Projection Pursuit

Eine dem mehrlagigen Perzeptron ähnliche statistische Methode ist zum Beispiel die sogenannte *Projection-Pursuit-Methode* (Friedman und Stuetzle 1981), welche man als

$$y = \sum_{h=1}^{H} \phi_h(\boldsymbol{w}_h^T \boldsymbol{x})$$

schreibt. Der Unterschied besteht darin, dass jede verborgene Einheit explizit ihre eigene separate Funktion $\phi_h(\cdot)$ besitzt, während im Fall des MLP oft alle verborgenen Einheiten die gleiche Funktion, etwa die Sigmoidfunktion, verwenden. In Kapitel 12 werden wir eine andere Struktur neuronaler Netze betrachten, welche als radiale Basisfunktion bezeichnet wird und die Gaußsche Funktion in den verborgenen Einheiten nutzt.

Es gibt viele Lehrbücher zum Thema künstliche neuronale Netze: Hertz, Krogh und Palmer 1991 – das früheste Werk – ist immer noch lesbar. Bishop 1995 setzt vor allem die Mustererkennung in den Mittelpunkt und diskutiert im Detail neben diversen Optimierungsalgorithmen, die für das Training verwendet werden können, auch den Bayesschen Ansatz

und generalisiert den Gewichtsabbau. Ripley 1996 analysiert neuronale Netze aus einer statistischen Perspektive heraus.

Künstliche neuronale Netze, zum Beispiel die mehrlagigen Perzeptronen, haben zahlreiche erfolgreiche Anwendungsgebiete. Zusätzlich zu ihren diversen effektiven Applikationen im Feld der adaptiven Steuerung, der Spracherkennung und visueller Anwendungen, sind zwei weitere nennenswert: das TD-Gammon-Programm von Tesauro verwendet bestärkendes Lernen (Kapitel 18), um ein mehrlagiges Perzeptron zu trainieren und spielt Backgammon auf höchster Stufe. Pomerleaus ALVINN ist ein neuronales Netz, welches selbstständig einen Lieferwagen bis circa 30 km/h schnell fährt, nachdem es gerade einmal fünf Minuten lang durch Beobachtung eines menschlichen Fahrers gelernt hat (Pomerleau 1991).

Die Forschung auf dem Gebiet der neuronalen Netze hat in den letzten Jahren starken Auftrieb erfahren, was insbesondere auf die Idee des tiefen Lernens und der tiefen neuronalen Netze zurückzuführen ist. Es ist zu beobachten, dass diese Verfahren in vielen Bereichen angewendet werden, beispielsweise in der Finanzwirtschaft, in der Biologie oder bei der natürlichen Sprachverarbeitung, und die Ergebnisse sind beeindruckend. Auf deeplearning.net finden Sie hierzu weitere Informationen. Dank wachsender Datenmengen und sinkender Hardwarekosten dürfte die Popularität dieser Anwendungen in naher Zukunft weiter zunehmen.

11.15 Übungen

1. Beschreiben Sie das Perzeptron, welches die NOT-Funktion aus seinen Eingaben berechnet.

   Lösung: $y = s(-x + 0{,}5)$

2. Beschreiben Sie das Perzeptron, welches die NAND-Funktion aus seinen zwei Eingaben berechnet.

3. Beschreiben Sie das Perzeptron, welches die Parität seiner drei Eingaben berechnet.

   Lösung:

   $$
   \begin{aligned}
   h_1 &= s(-x_1 - x_2 + 2x_3 - 1{,}5) \quad (001)\\
   h_2 &= s(-x_1 + 2x_2 - x_3 - 1{,}5) \quad (010)\\
   h_3 &= s(2x_1 - x_2 - x_3 - 1{,}5) \quad (100)\\
   h_4 &= s(x_1 + x_2 + x_3 - 2{,}5) \quad (111)\\
   y &= s(h_1 + h_2 + h_3 + h_4 - 0{,}5)
   \end{aligned}
   $$

   Die vier verborgenen Einheiten entsprechen den vier Fällen von (x_1, x_2, x_3)-Werten, für die die Parität 1 ist, nämlich 001, 010,

100 und 111. Wir verknüpfen sie dann mit OR, um die Gesamtausgabe zu berechnen. Eine andere Möglichkeit besteht darin, die drei-Bit-Parität mithilfe der zwei-Bit-Parität (XOR) als $(x_1 \text{ XOR } x_2) \text{ XOR } x_3$ zu berechnen.

4. Leiten Sie die Aktualisierungsgleichungen ab, wenn die verborgenen Einheiten die tanh-Funktion statt der Sigmoidfunktion verwenden. Nutzen Sie die Tatsache, dass $\tanh' = (1 - \tanh^2)$.

5. Leiten Sie die Aktualisierungsgleichungen für ein MLP mit zwei verborgenen Schichten ab.

   LÖSUNG: Definieren wir zunächst die Vorwärtsgleichungen:

$$z_{1h} = \text{sigmoid}(\boldsymbol{w}_{1h}^T \boldsymbol{x}) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), \; h = 1, \ldots, H_1$$

$$z_{2l} = \text{sigmoid}(\boldsymbol{w}_{2l}^T \boldsymbol{z}_1) = \text{sigmoid}\left(\sum_{h=0}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), \; l = 1, \ldots, H_2$$

$$y_i = \boldsymbol{v}_i^T \boldsymbol{z}_2 = \sum_{l=1}^{H_2} v_{il} z_{2l} + v_0 \; .$$

   Betrachten wir den Fall der Regression:

$$E = \frac{1}{2} \sum_t \sum_i \left(r_i^t - y_i^t\right)^2 .$$

   Wir führen eine Backpropagation aus, d. h., wir wenden weiter die Kettenregel an, und können den Fehler in einer Schicht als eine Funktion des Fehlers in der folgenden Schicht schreiben, wodurch der überwachte Fehler in der Ausgabeschicht in die vorherigen Schichten getragen wird:

$$err_i \equiv r_i^t - y_i^t \Rightarrow \Delta v_{il} = \eta \sum_t err_i z_{2l}$$

$$err_{2l} \equiv \left[\sum_i err_i v_i\right] z_{2l}(1 - z_{2l}) \Rightarrow \Delta w_{2lh} = \eta \sum_t err_{2l} z_{1h}$$

$$err_{1h} \equiv \left[\sum_l err_{2l} w_{2lh}\right] z_{1h}(1 - z_{1h}) \Rightarrow \Delta w_{1hj} = \eta \sum_t err_{1h} x_j \; .$$

6. Gegeben sei eine MLP-Architektur mit einer verborgenen Schicht, in der es auch direkte Gewichte von den Eingaben direkt zu den Ausgabeeinheiten gibt. Erläutern Sie, wann eine solche Struktur hilfreich ist und wie sie trainiert werden kann.

7. Die Parität ist invariant gegenüber zyklischen Verschiebungen, beispielsweise haben „0101“ und „1010“ die gleiche Parität. Entwerfen Sie ein mehrlagiges Perzeptron, um die Paritätsfunktion unter Nutzung dieses Hinweises zu lernen.

8. Worin bestehen bei der kaskadierenden Korrelation die Vorteile, wenn die vorher existierenden Gewichte festgehalten und nicht weiter verändert werden?

9. Leiten Sie die Aktualisierungsgleichungen für ein MLP ab, welches das Sammon-Mapping implementiert und den Sammon-Stress minimiert (Gleichung 11.40).

10. In Abschnitt 11.6 wird dargelegt, wie mit einem MLP eine stückweise konstante Approximation implementiert werden kann. Zeigen Sie, dass wir eine stückweise lineare Approximation implementieren können, wenn das Gewicht der letzten Schicht keine Konstante, sondern eine lineare Funktion der Eingabe ist.

11. Leiten Sie die Aktualisierungsgleichungen für das Soft Weight Sharing her.

    Lösung: Der Einfachheit halber nehmen wir ein einlagiges Netz für eine Klassifikation mit zwei Klassen an:

    $$y^t = \text{sigmoid}\left(\sum_i w_i x_i^t \right) .$$

    Der erweiterte Fehler ist

    $$E' = \log \sum_t r^t \log y^t + \lambda \sum_i \log \sum_{j=1}^{M} \alpha_j p_j(w_i)$$

    mit $p_j(w_i) \sim \mathcal{N}(m_j, s_j^2)$. Man beachte, dass $\{w_i\}_i$ alle Gewichte einschließlich der Verzerrung beinhaltet. Mithilfe des Gradientenabstiegs erhalten wir

    $$\Delta w_i^t = \eta(r^t - y^t)x_i^t - \eta\lambda \sum_j \pi_j(w_i) \frac{(w_i - m_j)}{s_j^2} .$$

    Dabei ist

    $$\pi_j(w_i) = \frac{\alpha_j p_j(w_i)}{\sum_l \alpha_l p_l(w_i)}$$

    die a-posteriori-Wahrscheinlichkeit, dass w_i zu Komponente j gehört. Das Gewicht wird aktualisiert, um die Kreuzentropie zu verringern und es gleichzeitig näher an den Mittelwert der nächsten

Gauß-Variable zu bringen. Mit einem solchen Schema können wir auch die Mischungsparameter aktualisieren, so zum Beispiel

$$\Delta m_j = \eta\lambda \sum_i \pi_j(w_i) \frac{(w_i - m_j)}{s_j^2} \; .$$

$\pi_j(w_i)$ liegt nahe bei 1, wenn es sehr wahrscheinlich ist, dass w_i aus der Komponente j stammt; in diesem Fall wird m_j aktualisiert, damit es näher an dem Gewicht w_i liegt, das es repräsentiert. Dies ist eine iterative Clusterprozedur, wie wir sie in Kapitel 12 ausführlicher diskutieren werden; siehe zum Beispiel Gleichung 12.5.

12. Wie können wir in einem Autoencoder-Netz über die Anzahl der verborgenen Einheiten entscheiden?

13. Das inkrementelle Lernen einer MLP-Struktur kann als Suche in einem Zustandsraum angesehen werden. Was sind hier die Operatoren? Was ist die Gütefunktion? Welcher Typ von Suchstrategie ist geeignet? Formulieren Sie die Antworten so, dass die dynamische Knotengenerierung und die kaskadierende Korrelation spezielle Instanzen sind.

14. Leiten Sie für das in Abbildung 11.20 gegebene MLP die Aktualisierungsgleichungen für das entfaltete Netz her.

11.16 Literaturangaben

Abu-Mostafa, Y. 1995. „Hints.“ *Neural Computation* 7: 639–671.

Aran, O., O. T. Yıldız, and E. Alpaydın. 2009. „An Incremental Framework Based on Cross-Validation for Estimating the Architecture of a Multilayer Perceptron.“ *International Journal of Pattern Recognition and Artificial Intelligence* 23: 159–190.

Ash, T. 1989. „Dynamic Node Creation in Backpropagation Networks.“ *Connection Science* 1: 365–375.

Battiti, R. 1992. „First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method.“ *Neural Computation* 4: 141–166.

Bengio, Y. 2009. „Learning Deep Architectures for AI.“ *Foundations and Trends in Machine Learning* 2 (1): 1–127.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Bourlard, H., and Y. Kamp. 1988. „Auto-Association by Multilayer Perceptrons and Singular Value Decomposition.“ *Biological Cybernetics* 59: 291–294.

Cottrell, G. W., P. Munro, and D. Zipser. 1987. „Learning Internal Representations from Gray-Scale Images: An Example of Extensional Programming.“ In *Ninth Annual Conference of the Cognitive Science Society,* 462–473. Hillsdale, NJ: Erlbaum.

Durbin, R., and D. E. Rumelhart. 1989. „Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks.“ *Neural Computation* 1: 133–142.

Fahlman, S. E., and C. Lebiere. 1990. „The Cascade Correlation Architecture.“ In *Advances in Neural Information Processing Systems 2,* ed. D. S. Touretzky, 524–532. San Francisco: Morgan Kaufmann.

Friedman, J. H., and W. Stuetzle. 1981. „Projection Pursuit Regression.“ *Journal of the American Statistical Association* 76: 817–823.

Fukushima, K. 1980. „Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.“ *Biological Cybernetics* 36: 193–202.

Geman, S., E. Bienenstock, and R. Doursat. 1992. „Neural Networks and the Bias/Variance Dilemma.“ *Neural Computation* 4: 1–58.

Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation.* Reading, MA: Addison Wesley.

Hinton, G. E., and R. R. Salakhutdinov. 2006. „Reducing the dimensionality of data with neural networks.“ *Science* 313: 504–507.

Hopfield, J. J. 1982. „Neural Networks and Physical Systems with Emergent Collective Computational Abilities.“ *Proceedings of the National Academy of Sciences USA* 79: 2554–2558.

Hornik, K., M. Stinchcombe, and H. White. 1989. „Multilayer Feedforward Networks Are Universal Approximators.“ *Neural Networks* 2: 359–366.

Le Cun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. „Backpropagation Applied to Handwritten Zipcode Recognition.“ *Neural Computation* 1: 541–551.

MacKay, D. J. C. 1992a. „Bayesian Interpolation.“ *Neural Computation* 4: 415–447.

MacKay, D. J. C. 1992b. „A Practical Bayesian Framework for Backpropagation Networks“ *Neural Computation* 4: 448–472.

Mao, J., and A. K. Jain. 1995. „Artificial Neural Networks for Feature Extraction and Multivariate Data Projection.“ *IEEE Transactions on Neural Networks* 6: 296–317.

Marr, D. 1982. *Vision.* New York: Freeman.

McCulloch, W. S., and W. Pitts. 1943. „A Logical Calculus of the Ideas Immenent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5: 115–133.

Minsky, M. L., and S. A. Papert. 1969. *Perceptrons.* Cambridge, MA: The MIT Press. (Expanded ed. 1990.)

Nowlan, S. J., and G. E. Hinton. 1992. „Simplifying Neural Networks by Soft Weight Sharing." *Neural Computation* 4: 473–493.

Pomerleau, D. A. 1991. „Efficient Training of Artificial Neural Networks for Autonomous Navigation." *Neural Computation* 3: 88–97.

Posner, M. I., ed. 1989. *Foundations of Cognitive Science.* Cambridge, MA: The MIT Press.

Richard, M. D., and R. P. Lippmann. 1991. „Neural Network Classifiers Estimate Bayesian *a Posteriori* Probabilities." *Neural Computation* 3: 461–483.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks.* Cambridge, UK: Cambridge University Press.

Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* New York: Spartan.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986a. „Learning Representations by Backpropagating Errors." *Nature* 323: 533–536.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986b. „Learning Internal Representations by Error Propagation." In *Parallel Distributed Processing,* ed. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, 318–362. Cambridge, MA: The MIT Press.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, eds. 1986. *Parallel Distributed Processing.* Cambridge, MA: The MIT Press.

Simard, P., B. Victorri, Y. Le Cun, and J. Denker. 1992. „Tangent Prop: A Formalism for Specifying Selected Invariances in an Adaptive Network." In *Advances in Neural Information Processing Systems 4,* ed. J. E. Moody, S. J. Hanson, R. P. Lippman, 895–903. San Francisco: Morgan Kaufmann.

Tesauro, G. 1994. „TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play." *Neural Computation* 6: 215–219.

Thagard, P. 2005. *Mind: Introduction to Cognitive Science.* 2nd ed. Cambridge, MA: MIT Press.

Waibel, A., T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. 1989. „Phoneme Recognition Using Time-Delay Neural Networks.“ *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37: 328–339.

Williams, R. J., and D. Zipser. 1989. „A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.“ *Neural Computation* 1: 270–280.

12 Lokale Modelle

Wir setzen unsere Betrachtungen von mehrlagigen neuronalen Netzen mit Modellen fort, bei denen die erste Schicht lokal rezeptive Einheiten enthält, welche auf Instanzen in einer lokalisierten Region des Eingaberaums reagieren. Die darauf folgende zweite Schicht erlernt die Regressions- oder Klassifikationsfunktion für diese lokalen Regionen. Wir diskutieren Lernmethoden für das Finden von wichtigen lokalen Regionen sowie die dort verantwortlichen Modelle.

12.1 Einführung

Eine Möglichkeit zur Approximation von Funktionen besteht darin, den Eingaberaum in lokale Teilstücke zu zerlegen und eine separate Anpassung in jedem lokalen Teilstück zu lernen. In Kapitel 7 haben wir statistische Clustermethoden diskutiert, die es uns ermöglichten, Eingabeinstanzen zu gruppieren und die Eingabeverteilung zu modellieren. Kompetitive Methoden sind Methoden neuronaler Netze, die bei der online Clustermethode zum Einsatz kommen. In diesem Kapitel befassen wir uns mit der online Version des k-Means-Algorithmus sowie mit zwei Erweiterungen für ein neuronales Netz, nämlich der Adaptiven Resonanztheorie (ART) und der selbstorganisierenden Merkmalskarte (engl. *self-organizing map*, SOM).

Danach betrachten wir, wie überwachtes Lernen implementiert wird, sobald die Eingaben einmal lokalisiert sind. Ist die Anpassung in einem lokalen Teilstück konstant, so bezeichnet man die Technik als Netz radialer Basisfunktionen (RBF); ist sie eine lineare Funktion der Eingabe, so spricht man von gemischten Expertensystemen (engl. *mixture of experts*, MoE). Wir diskutieren sowohl die Regression als auch die Klassifikation und vergleichen diesen Ansatz außerdem mit dem MLP, das wir eben in Kapitel 11 behandelt haben.

12.2 Kompetitives Lernen

In Kapitel 7 haben wir die semiparametrische gemischte Gauß-Verteilung genutzt, welche annimmt, dass die Eingabe aus einer von k Gaußschen Quellen entstammt. In diesem Abschnitt treffen wir dieselbe Annahme,

also dass k Gruppen (oder Cluster) in den Daten existieren, jedoch ist unser Vorgehen insofern nicht probabilistisch, als dass wir kein parametrisches Modell für die Quellen erzwingen. Ein weiterer Unterschied besteht darin, dass die von uns vorgeschlagenen Lernmethoden online Methoden sind: uns liegt also während der Trainingsphase nicht die komplette Stichprobe vor, sondern wir erhalten die Instanzen eine nach der anderen und aktualisieren mit jedem Erhalt die Modellparameter. Die Bezeichung *kompetitives Lernen* oder *Wettbewerbslernen* wird genutzt, da es scheint, als ob diese Gruppen, oder genauer gesagt, die sie repräsentierenden Einheiten, miteinander darum in Konkurrenz stehen, wer eine Instanz repräsentieren darf. Das Modell wird auch *Winner-Take-All-Modell* genannt; es ist, als ob eine Gruppe gewinnt und aktualisiert wird, während die anderen überhaupt keine Aktualisierung zugewiesen bekommen.

KOMPETITIVES LERNEN

WINNER-TAKE-ALL-MODELL

Diese Methoden können für sich genommen für das online Clusterverfahren genutzt werden, im Gegensatz zu den in Kapitel 7 diskutierten Batch-Methoden. Eine online Methode hat die üblichen Vorteile: (1) wir brauchen keinen zusätzlichen Speicherplatz, um den kompletten Trainingsdatensatz zu speichern, (2) Aktualisierungen sind in jedem Schritt leicht zu implementieren, beispielsweise in der Hardware und (3) die Eingabeverteilung kann sich zeitlich ändern und das Modell passt sich selbst automatisch diesen Veränderungen an. Würden wir einen Batch-Algorithmus verwenden, so müssten wir eine neue Stichprobe erfassen und die Batchmethode erneut über die gesamte Stichprobe laufen lassen.

Ab Abschnitt 12.3 werden wir uns außerdem damit befassen, wie ein Ansatz solcher durch eine überwachte Methode verfolgt werden kann, um Regressions- oder Klassifikationsprobleme zu lernen. Hierbei wird es sich um ein zweistufiges System handeln, welches durch ein zweischichtiges Netz implementiert werden kann, wobei die erste Stufe (Schicht) die Eingabedichte modelliert und das verantwortliche lokale Modell bestimmt und die zweite Stufe die Generierung der finalen Ausgabe durch das lokale Modell enthält.

12.2.1 Online k-Means-Algorithmus

In Gleichung 7.3 definierten wir den Rekonstruktionsfehler als

$$E(\{\boldsymbol{m}_i\}_{i=1}^k|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 , \tag{12.1}$$

mit

$$b_i^t = \begin{cases} 1 \text{ falls } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_l \|\boldsymbol{x}^t - \boldsymbol{m}_l\| , \\ 0 \text{ andernfalls} . \end{cases} \tag{12.2}$$

$\mathcal{X} = \{\boldsymbol{x}^t\}_t$ ist die Stichprobe und $\boldsymbol{m}_i, i = 1, \ldots, k$ sind die Zentren der Cluster. b_i^t ist 1, falls $\boldsymbol{m}_i$ das nächstliegende Zentrum zu $\boldsymbol{x}^t$ gemessen

durch die Euklidische Distanz ist. Es scheint, als ob alle $\boldsymbol{m}_l, l = 1, \ldots, k$ miteinander konkurrieren und $\boldsymbol{m}_i$ den Wettbewerb „gewinnt“, da es am nächsten liegt.

Der Batch-Algorithmus k-Means aktualisiert die Zentren als

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t} , \tag{12.3}$$

wodurch Gleichung 12.1 minimiert wird, sobald die „Gewinner“ mittels Gleichung 12.2 ausgewählt wurden. Wie wir bereits gesehen haben, werden diese zwei Schritte der Berechnung von b_i^t und der Aktualisierung von $\boldsymbol{m}_i$ bis zur Konvergenz wiederholt.

ONLINE k-MEANS

Wir können den *online k-Means* durch die Anwendung eines stochastischen Gradientenabstiegs beschreiben, wobei die Instanzen eine nach der anderen betrachtet werden und bei jedem Schritt eine kleine Aktualisierung eingebracht wird, ohne dabei den Effekt der vorhergehenden Aktualisierungen zu vergessen. Der Rekonstruktionsfehler für eine einzelne Instanz ist

$$\begin{aligned} E^t(\{\boldsymbol{m}_i\}_{i=1}^k | \boldsymbol{x}^t) &= \frac{1}{2} \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 \\ &= \frac{1}{2} \sum_i \sum_{j=1}^d b_i^t (x_j^t - m_{ij})^2 , \end{aligned} \tag{12.4}$$

wobei b_i^t wie in Gleichung 12.2 definiert ist. Wenden wir darauf den Gradientenabstieg an, so erhalten wir die folgende Aktualisierungsregel für jede Instanz $\boldsymbol{x}^t$:

$$\Delta m_{ij} = -\eta \frac{\partial E^t}{\partial m_{ij}} = \eta b_i^t (x_j^t - m_{ij}) . \tag{12.5}$$

Dadurch wird das nächstliegende Zentrum (für welches $b_i^t = 1$) in Richtung der Eingabe um einen durch η gegebenen Faktor verschoben. Die anderen Zentren haben $b_l^t, l \neq i$ gleich 0 und werden nicht aktualisiert (siehe Abbildung 12.1). Eine Batch-Prozedur kann ebenfalls definiert werden, indem Gleichung 12.5 über alle t hinweg summiert wird. Wie bei jedem Gradientenabstiegsverfahren kann auch hier ein Momentumsterm hinzugefügt werden. Um Konvergenz zu erreichen, wird η schrittweise auf 0 herunter gesenkt. Doch dies impliziert das *Dilemma von Stabilität und Plastizität*: wird η in Richtung 0 verringert, wird das Netz stabiler, jedoch verlieren wir die Anpassungsfähigkeit an neue Muster, die im Laufe der Zeit auftauchen können, da Aktualisierungen zu klein werden. Behalten wir für η einen hohen Wert, kann dies zu Oszillationen von $\boldsymbol{m}_i$ führen.

DILEMMA VON STABILITÄT UND PLASTIZITÄT

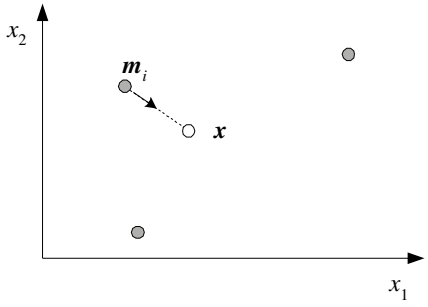


Abb. 12.1: Die schattierten Kreise sind die Zentren und der leere Kreis steht für die Eingabeinstanz. Die online Version des k-Means verschiebt das nächstliegende Zentrum entlang der Ausrichtung von $(\boldsymbol{x} - \boldsymbol{m}_i)$ um einen durch η spezifizierten Faktor.

Der Pseudocode des online k-Means ist im Listing 12.1 dargestellt. Hierbei handelt es sich um die online Version des im Listing 7.1 gegebenen Batch-Algorithmus.

```
Initialisiere m_i, i = 1, ..., k, zum Beispiel mit k zufälligen x^t
Repeat
    For alle x^t ∈ X in zufälliger Reihenfolge
        i ← argmin_j ||x^t − m_j||
        m_i ← m_i + η(x^t − m_j)
Until m_i konvergieren
```

Listing 12.1: Online k-Means-Algorithmus. Die Batch-Version ist im Listing 7.1 dargestellt.

Das kompetitive Netz kann als ein einschichtiges rekurrentes Netz implementiert werden, so wie in Abbildung 12.2 dargestellt. Die Eingabeschicht enthält den Eingabevektor $\boldsymbol{x}$; man beachte, dass es keine Verzerrungseinheit gibt. Die Werte der Ausgabeeinheiten sind die b_i, und bei ihnen handelt es sich um Perzeptronen:

$$b_i = \boldsymbol{m}_i^T \boldsymbol{x} \,. \tag{12.6}$$

Sodann müssen wir das Maximum von b_i wählen und es gleich 1 setzen; die anderen, $b_l, l \neq i$, sind gleich 0 zu setzen. Wenn wir alles völlig neutral durchführen wollen, das heißt, ein Netz von gleichzeitig operierenden Verarbeitungseinheiten verwenden wollen, kann die Wahl des Maximums durch *laterale Inhibition* implementiert werden. Wie in Abbildung 12.2 gezeigt, hat jede Einheit eine exzitatorische rekurrente Verbindung (d.h. mit einem positiven Gewicht) mit sich selbst und inhibierenden, rekurrenten Verbindungen (d.h. mit negativen Gewichten) zu den anderen Ausgabeeinheiten. Mit einer entsprechenden nichtlinearen Aktivierungsfunktion und positiven bzw. negativen rekurrenten Gewichtswerten konvergiert solch ein Netzwerk nach einigen Iterationen zu einem Zustand, in dem das Maximum gleich 1 ist und alle anderen Werte gleich 0 sind (Grossberg 1980; Feldman und Ballard 1982).

LATERALE INHIBITION

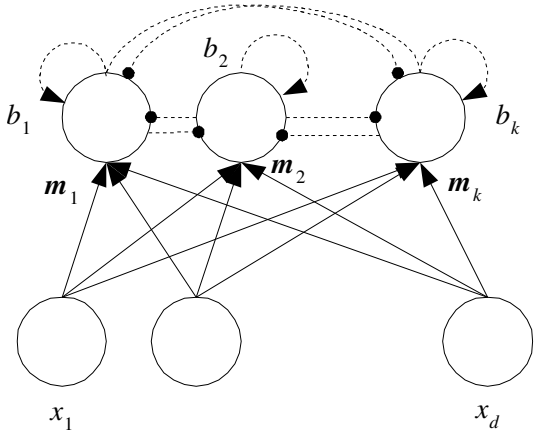


***Abb. 12.2*:** Das kompetitive neuronale Netz nach dem Winner-Take-All-Prinzip; es handelt sich um ein Netz mit k Perzeptronen mit rekurrenten Verbindungen an der Ausgabe. Die gestrichelten Linien sind die rekurrenten Verbindungen, von denen die mit einem Pfeil am Ende erregend (exzitatorisch) und die mit einem Kreis hemmend (inhibitorisch) sind. Jede Einheit an der Ausgabe bestärkt ihren Wert und versucht, andere Ausgaben zu unterdrücken. Bei einer entsprechenden Verteilung von diesen rekurrenten Gewichten unterdrückt das Maximum alle Anderen. Das bewirkt den Effekt, dass die Einheit, deren $\boldsymbol{m}_i$ am nahesten an x liegt, am Ende des Trainings einen Wert von 1 für ihr b_i erhält. Alle anderen Einheiten erhalten ein b_l mit $l \neq i$ gleich null.

Das in Gleichung 12.6 genutzte Skalarprodukt ist ein Ähnlichkeitsmaß; wir haben bereits in Abschnitt 5.5 gesehen (Gleichung 5.26), dass bei gleichen Betragswerten für $\boldsymbol{m}_i$ dann die Einheit mit der minimalen Euklidischen Distanz, $\|\boldsymbol{m}_i - \boldsymbol{x}\|$, dieselbe ist, wie diejenige mit dem maximalen Skalarprodukt $\boldsymbol{m}_i^T \boldsymbol{x}$.

An dieser Stelle und auch später, wenn wir andere kompetitive Methoden diskutieren werden, nutzen wir die Euklidische Distanz, sollten jedoch immer bedenken, dass die Verwendung der Euklidischen Distanz impliziert, dass alle Eingabeattribute die gleiche Varianz besitzen und dass sie nicht korrelieren. Ist dies nicht der Fall, so sollte sich das im Distanzmaß widerspiegeln; das heißt, der Mahalanobis-Abstand sollte verwendet werden oder eine angemessene Normalisierung, beispielsweise durch die PCA, ist in einer Vorbearbeitungsstufe durchzuführen, bevor die Euklidische Distanz zum Einsatz kommt.

Wir können dann Gleichung 12.5 umschreiben als

$$\Delta m_{ij}^t = \eta b_i^t x_j^t - \eta b_i^t m_{ij} \; . \tag{12.7}$$

Es sei daran erinnert, dass m_{ij} das Gewicht der Verbindung von x_j zu b_i ist. Eine Aktualisierung der Form, wie wir im ersten Term

$$\Delta m_{ij}^t = \eta b_i^t x_j^t \tag{12.8}$$

Hebbsches Lernen

sehen, nennt man *Hebbsches Lernen* und definiert damit die Aktualisierung als das Produkt der Werte der präsynaptischen und postsynaptischen Einheiten. Es wurde als Modell zu Zwecken neuronaler Plastizität entwickelt: eine Synapse gewinnt an Bedeutung, wenn sich die Einheiten vor und nach der Verbindung simultan entladen und dadurch andeuten, dass sie miteinander korrelieren. Beim Hebbschen Lernen alleine wachsen die Gewichte jedoch ohne Beschränkung ($x_j^t \geq 0$) und wir benötigen eine zweite Kraft, um die Gewichte, die nicht aktualisiert werden, zu senken. Eine Möglichkeit besteht darin, die Gewichte explizit zu normalisieren, so dass $\|\boldsymbol{m}_i\| = 1$; falls $\Delta m_{ij} > 0$ und $\Delta m_{il} = 0, l \neq i$, so verringern sich die m_{il}, sobald wir $\boldsymbol{m}_i$ auf Einheitslänge normalisieren. Eine andere Möglichkeit bietet die Einführung eines Terms zum Gewichtsabbau (Oja 1982), wobei der zweite Term aus Gleichung 12.7 als solcher aufgefasst werden kann. Hertz, Krogh und Palmer (1991) behandeln kompetitive Netze sowie Hebbsches Lernen im Detail und beweisen beispielsweise, wie solche Netze lernen können, eine PCA durchzuführen. Mao und Jain (1995) widmen sich online Algorithmen für die PCA und LDA.

Wie wir in Kapitel 7 gesehen haben, besteht ein Problem darin, tote Zentren zu vermeiden, also diejenigen, die existieren aber nicht effektiv genutzt werden. Im Fall von kompetitiven Netzen entspricht dies den Zentren, die niemals den Wettbewerb gewinnen, weil sie weit entfernt von der Eingabe initialisiert werden. Es gibt verschiedene Mittel, um dies zu vermeiden:

1. Wir können die $\boldsymbol{m}_i$ durch zufällig gewählte Eingabeinstanzen initialisieren und sicherstellen, dass sie von einem Punkt aus beginnen, an dem Daten existieren.

2. Wir können einen vorgeschalteten Cluster-Algorithmus verwenden und Einheiten eine nach der anderen hinzufügen, wobei sie immer an einer Stelle eingefügt werden, an der Bedarf für sie besteht. Ein Beispiel ist das ART-Modell, welches wir in Abschnitt 12.2.2 behandeln werden.

3. Wenn wir aktualisieren, so tun wir dies nicht nur für das Zentrum der nächstliegenden Einheit, sondern auch für einige andere Zentren. Da diese aktualisiert werden, bewegen sie sich auch in Richtung der Eingabe, d.h. sie bewegen sich schrittweise in Richtung der Ausschnitte des Eingaberaums, in denen Eingaben existieren, und gewinnen möglicherweise den Wettbewerb. Ein Beispiel, welches wir in Abschnitt 12.2.3 betrachten werden, ist die SOM.

4. Eine weitere Möglichkeit besteht darin, einen *Gewissensmechanismus* (DeSieno 1988) einzuführen: eine Einheit, die vor kurzem den Wettbewerb gewonnen hat, fühlt sich quasi „schuldig“ und lässt andere gewinnen.

12.2.2 Adaptive Resonanztheorie

Die Anzahl an Gruppen k sollte bekannt und spezifiziert sein, bevor die Parameter berechnet werden können. Ein weiterer Ansatz ist der *inkrementelle*, bei dem man mit einer einzelnen Gruppe beginnt und neue Gruppen je nach Bedarf hinzufügt. Wir diskutieren den Algorithmus der *adaptiven Resonanztheorie* (ART) (Carpenter und Grossberg 1988) als ein Beispiel für inkrementelle Algorithmen. Beim ART berechnen bei vorliegender Eingabe alle Ausgabeeinheiten ihre Werte und diejenige, die der Eingabe am nächsten ist, wird ausgewählt. Hierbei handelt es sich entweder um die Einheit mit dem Maximalwert, wenn die Einheit das Skalarprodukt nutzt wie in Gleichung 12.6, oder um die mit dem Minimalwert, wenn die Einheit die Euklidische Distanz verwendet.

ADAPTIVE RESONANZTHEORIE

Nehmen wir einmal an, wir nutzen die Euklidische Distanz. Ist der Minimalwert kleiner als ein gewisser Schwellwert, auch als *Vigilanz* bezeichnet, dann wird die Aktualisierung wie beim k-Means eingebracht. Ist die Distanz größer als die Vigilanz, so wird eine neue Ausgabeeinheit hinzugefügt und ihr Zentrum wird mit der Instanz initialisiert. Dadurch wird eine Hypersphäre definiert, deren Radius durch die Vigilanz gegeben ist, wodurch das Umfangsvolumen jeder Einheit definiert wird. Wir fügen immer dann eine neue Einheit hinzu, wenn uns eine Eingabe vorliegt, die nicht durch eine der Einheiten abgedeckt wird (siehe Abbildung 12.3).

VIGILANZ

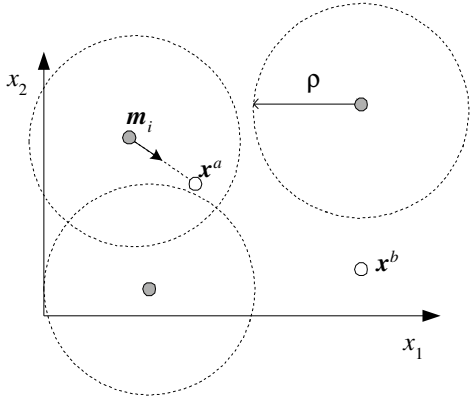


Abb. 12.3: Die Entfernung von $\boldsymbol{x}^a$ zum nächstliegenden Zentrum ist kleiner als der Vigilanzwert ρ und das Zentrum wird wie beim online k-Means aktualisiert. Jedoch ist $\boldsymbol{x}^b$ nicht nahe genug an irgendeinem der Zentren und eine neue Gruppe sollte an jener Position erstellt werden.

Bezeichne ρ die Vigilanz, so nutzen wir die folgenden Gleichungen bei jeder Aktualisierung:

$$b_i = \|\boldsymbol{m}_i - \boldsymbol{x}^t\| = \min_{l=1}^{k} \|\boldsymbol{m}_l - \boldsymbol{x}^t\| , \tag{12.9}$$

$$\begin{cases} \boldsymbol{m}_{k+1} \leftarrow \boldsymbol{x}^t & \text{falls } b_i > \rho , \\ \Delta\boldsymbol{m}_i = \eta(\boldsymbol{x}^t - \boldsymbol{m}_i) & \text{andernfalls} . \end{cases}$$

Das Einfügen eines Schwellwertes für die Distanz entspricht der Festlegung eines Schwellwertes für den Rekonstruktionsfehler pro Instanz. Handelt es sich um eine Euklidische Distanz und ist der Fehler wie in Gleichung 12.4 definiert, so weist dies darauf hin, dass der maximal pro Instanz erlaubte Rekonstruktionsfehler gleich dem Quadrat der Vigilanz ist.

12.2.3 Selbstorganisierende Merkmalskarten

Eine Möglichkeit, das Vorhandensein toter Einheiten zu vermeiden, besteht darin, nicht nur die Gewinnereinheit, sondern auch einige der anderen Einheiten zu aktualisieren. Bei der von Kohonen (1990, 1995) vorgeschlagenen *selbstorganisierenden Merkmalskarte* (engl. self-organizing map, SOM) definieren Indexwerte für Einheiten, sprich i wie in $\boldsymbol{m}_i$, eine *Nachbarschaft* für die Einheiten. Ist $\boldsymbol{m}_i$ am nächsten zum Zentrum, so werden zusätzlich zu $\boldsymbol{m}_i$ auch seine Nachbarn aktualisiert. Ist die Nachbarschaft zum Beispiel von der Größe 2, so werden auch $\boldsymbol{m}_{i-2}, \boldsymbol{m}_{i-1}, \boldsymbol{m}_{i+1}, \boldsymbol{m}_{i+2}$ aktualisiert, jedoch mit geringerem Gewicht, je mehr sich die Nachbarschaft ausweitet. Ist i der Index für das nächstliegende Zentrum, so werden die Zentren aktualisiert mit

SELBSTORGANISIERENDE MERKMALSKARTE

$$\Delta\boldsymbol{m}_l = \eta\ e(l,i)(\boldsymbol{x}^t - \boldsymbol{m}_l) , \tag{12.10}$$

wobei $e(l,i)$ der *Nachbarschaftsfunktion* entspricht. $e(l,i) = 1$, wenn $l = i$, und der Wert nimmt ab, wenn $|l - i|$ zunimmt, beispielsweise als Gauß-Verteilung, $\mathcal{N}(i,\sigma)$:

$$e(l,i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(l-i)^2}{2\sigma^2}\right] . \tag{12.11}$$

Zu Zwecken der Konvergenz verringert sich der Support der Nachbarschaftsfunktion mit der Zeit, beispielsweise nimmt σ ab, und am Ende wird nur die Gewinnereinheit aktualisiert.

Da die benachbarten Einheiten auch in Richtung Eingabe verschoben werden, vermeiden wir tote Einheiten, weil sie den Wettbewerb zu einem späteren Zeitpunkt gewinnen werden, nach ein wenig Anschubhilfe von ihren benachbarten Freunden (siehe Abbildung 12.4).

Die Aktualisierung der Nachbarn hat den Effekt, dass selbst bei zufälliger Initialisierung der Zentren aufgrund deren gemeinsamer Bewegung in

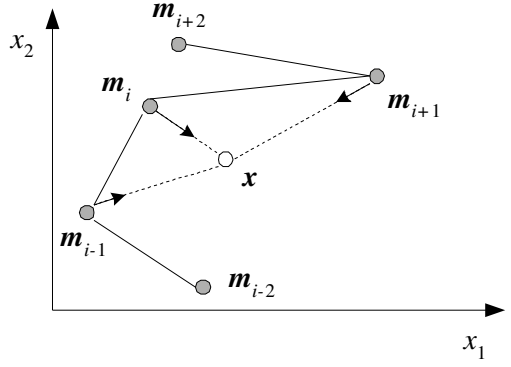


Abb. 12.4: Bei der SOM werden nicht nur die nächstliegende Einheit, sondern auch deren durch Indexwerte bestimmte Nachbarn in Richtung der Eingabe verschoben. Hier ist die Nachbarschaft gleich 1; $\boldsymbol{m}_i$ und seine 1-nächsten Nachbarn werden aktualisiert. Man beachte, dass hier $\boldsymbol{m}_{i+1}$ zwar weit entfernt von $\boldsymbol{m}_i$ liegt. Doch weil es zusammen mit $\boldsymbol{m}_i$ aktualisiert wird, und weil $\boldsymbol{m}_i$ aktualisiert wird, wenn $\boldsymbol{m}_{i+1}$ der Sieger ist, werden die beiden im Eingaberaum ebenfalls Nachbarn sein.

Richtung derselben Eingabe Einheiten mit benachbartem Index auch Nachbarn im Eingaberaum sein werden, sobald das System konvergiert.

Bei den meisten Anwendungen sind die Einheiten in Form einer zweidimensionalen *Karte* organisiert. Das heißt, jede Einheit hat zwei Indexwerte, $\boldsymbol{m}_{i,j}$, und die Nachbarschaft wird in zwei Dimensionen definiert. Falls $\boldsymbol{m}_{i,j}$ das nächstliegende Zentrum ist, so werden die Zentren aktualisiert mit

$$\Delta \boldsymbol{m}_{k,l} = \eta e(k,l,i,j)(\boldsymbol{x}^t - \boldsymbol{m}_{k,l}) \,, \tag{12.12}$$

wobei die Nachbarschaftsfunktion nun in zwei Dimensionen liegt. Nach der Konvergenz entsteht daraus eine zweidimensionale *topographische Karte* des originalen d-dimensionalen Eingaberaums. Die Karte enthält viele Einheiten in Teilen des Raums, in denen die Dichte groß ist, und keine Einheit wird Teilen zugewiesen, in denen sich keine Eingabe findet. Sobald die Karte konvergiert, werden Eingaben, die im Originalraum nah beieinander liegen, auf Einheiten abgebildet, die in der Karte dicht zusammen liegen. In dieser Hinsicht kann die Karte als eine nichtlineare Form der multidimensionalen Skalierung interpretiert werden, wobei aus dem originalen $\boldsymbol{x}$-Raum in die zwei Dimensionen (i,j) abgebildet wird. Ähnlich dazu werden die Einheiten bei einer eindimensionalen Karte auf die Kurve der größten Dichte im Eingaberaum platziert, und zwar als eine *Hauptkurve*.

TOPOGRAPHISCHE KARTE

12.3 Radiale Basisfunktionen

In einem mehrlagigen Perzeptron (Kapitel 11), bei dem verborgene Einheiten das Skalarprodukt nutzen, definiert jede verborgene Einheit eine Hyperebene, und durch die sigmoidale Nichtlinearität hat eine verborgene Einheit einen Wert zwischen 0 und 1, wodurch die Position der Instanz im Bezug zur Hyperebene codiert wird. Jede Hyperebene zerlegt den Eingaberaum in zwei Teile, und typischerweise haben für eine gegebene Eingabe viele der verborgenen Einheiten Ausgaben ungleich 0. Dies bezeichnet man als *verteilte Repräsentation*, weil die Eingabe durch die simultane Aktivierung von vielen verborgenen Einheiten codiert wird.

VERTEILTE REPRÄSENTATION

Eine weitere Möglichkeit besteht darin, eine *lokale Repräsentation* zu nutzen, wobei für eine gegebene Eingabe nur eine oder einige wenige Einheiten aktiv sind. Es ist, als ob diese *lokal abgestimmten Einheiten* den Eingaberaum unter sich selbst partitionieren und nur hinsichtlich gewisser Eingaben selektiv sind. Der Teil des Eingaberaums, in dem eine Einheit eine Reaktion ungleich 0 besitzt, wird als *rezeptives Feld* bezeichnet. Der Eingaberaum wird dann durch Einheiten abgedeckt.

LOKALE REPRÄSENTATION

REZEPTIVES FELD

Neuronen mit derlei Reaktionscharakteristika finden sich in vielen Teilen des Kortex. Beispielsweise reagieren Zellen im visuellen Kortex selektiv auf eine Stimulierung, die lokal sowohl hinsichtlich der retinalen Position als auch des Winkels der visuellen Orientierung ist. Solche lokal abgestimmten Zellen sind typischerweise in topographischen kortikalen Karten arrangiert, in denen die Werte der Variablen, auf welche die Zellen reagieren, in ihrer Position in der Karte variieren können, so wie bei einer SOM.

Das Konzept von Lokalität impliziert eine Distanzfunktion für die Messung der Ähnlichkeit zwischen der gegebenen Eingabe $\boldsymbol{x}$ und der Position der Einheit h, $\boldsymbol{m}_h$. Häufig wird dieses Maß als Euklidische Distanz gewählt, $\|\boldsymbol{x} - \boldsymbol{m}_h\|$. Die Reaktionsfunktion wird so gesetzt, dass sie ein Maximum an der Stelle mit $\boldsymbol{x} = \boldsymbol{m}_h$ besitzt und abnimmt, je unähnlicher die beiden Werte werden. Üblicherweise nutzen wir die Gaußsche Funktion (siehe Abbildung 12.5):

$$p_h^t = \exp\left[-\frac{\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2}{2s_h^2}\right] . \tag{12.13}$$

Streng genommen handelt es sich nicht um eine Gauß-Verteilung, aber wir verwenden dennoch die gleiche Bezeichnung. $\boldsymbol{m}_j$ beziehungsweise s_j bezeichnen das Zentrum und die Streuung der lokalen Einheit j, und definieren somit eine radial symmetrische Basisfunktion. Man kann eine elliptische verwenden mit verschiedenen Streuungen in unterschiedlichen Dimensionen, oder man nutzt gar den kompletten Mahalanobis-Abstand, um korrelierende Eingaben zu erlauben; das hat allerdings den Preis, dass das Modell komplizierter wird (Übung 2).

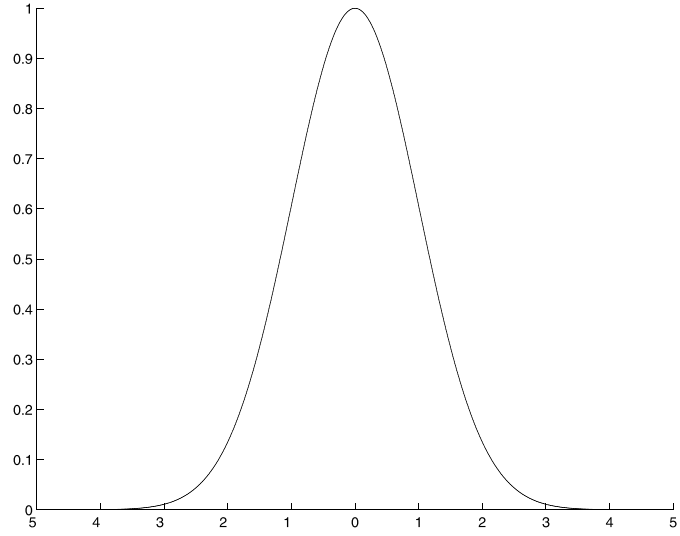
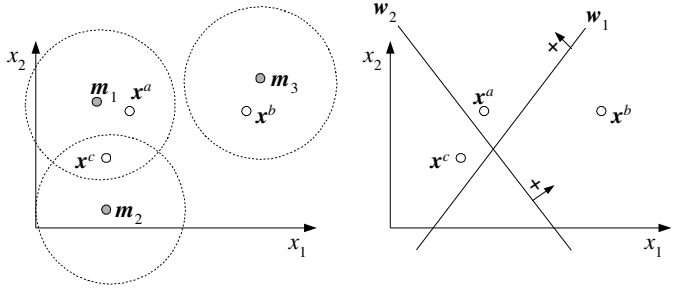


***Abb. 12.5*:** Die eindimensionale Form einer glockenförmigen Funktion, die im Netz radialer Basisfunktionen verwendet wird. Hier ist $m = 0$ und $s = 1$. Sie ähnelt einer Gaußschen Dichte, ist aber keine Dichte; sie integriert nicht zu 1. Sie ist ungleich 0 zwischen $(m - 3s, m + 3s)$, aber ein konservativeres Intervall ist $(m - 2s, m + 2s)$.

Die Idee hinter der Verwendung solcher lokalen Basisfunktionen ist die, dass in den Eingabedaten Gruppen oder Cluster von Instanzen existieren und für jedes solches Cluster definieren wir eine Basisfunktion p_h^t, die ungleich 0 ist, wenn die Instanz $\boldsymbol{x}^t$ zum Cluster h gehört. Jede der online kompetitiven Methoden, die in Abschnitt 12.2 besprochen wurden, kann hier angewandt werden, um die Zentren $\boldsymbol{m}_h$ zu finden. Es existiert eine einfache und effektive Heuristik, um die Streuungen zu bestimmen: sobald wir die Zentren herausgefunden haben, bestimmen wir für jedes Cluster die entfernteste durch das Cluster abgedeckte Instanz und legen für s_h die Hälfte ihrer Distanz zum Zentrum als Wert fest. Wir hätten auch ein Drittel nutzen können, aber hier ziehen wir eine konservative Vorgehensweise vor. Wir können auch die statistische Clustermethode nutzen, beispielsweise den EM-Algorithmus bei gemischten Gauß-Verteilungen, die wir in Kapitel 7 besprochen haben, um die Clusterparameter zu bestimmen, sprich Mittelwerte und Varianzen (und Kovarianzen).

$p_h^t, h = 1, \ldots, H$ definieren einen neuen H-dimensionalen Raum und formen eine neue Repräsentation von $\boldsymbol{x}^t$. Wir können auch b_h^t (Gleichung 12.2) nutzen, um die Eingaben zu codieren, aber b_h^t sind gleich 0/1; p_h^t haben den zusätzlichen Vorteil, dass sie die Entfernung zu ihrem Zentrum durch einen Wert in $(0, 1)$ codieren. Die Geschwindigkeit, mit der dieser Wert auf 0 abfällt, hängt von s_h ab. Abbildung 12.6 zeigt ein



Lokale Repräsentation im Raum von (p_1, p_2, p_3)

$\boldsymbol{x}^a$: (1.0, 0.0, 0.0)
$\boldsymbol{x}^b$: (0.0, 0.0, 1.0)
$\boldsymbol{x}^c$: (1.0, 1.0, 0.0)

Verteilte Repräsentation im Raum von (h_1, h_2)

$\boldsymbol{x}^a$: (1.0, 1.0)
$\boldsymbol{x}^b$: (0.0, 1.0)
$\boldsymbol{x}^c$: (1.0, 0.0)

***Abb. 12.6*:** Der Unterschied zwischen lokalen und verteilten Repräsentationen. Die Werte sind harte 0/1-Werte. Man kann weiche Werte in $(0, 1)$ nutzen und eine informationshaltigere Codierung erzielen. In der lokalen Repräsentation wird dies durch die Gaußsche RBF getan, welche die Distanz zum Zentrum, $\boldsymbol{m}_i$, nutzt; bei der verteilten Repräsentation geschieht dies durch die Sigmoidfunktion, welche die Distanz zur Hyperebene, $\boldsymbol{w}_i$, verwendet.

LOKALE VS. VERTEILTE REPRÄSENTATION

Beispiel und vergleicht so eine *lokale Repräsentation* mit einer *verteilten Repräsentation,* wie sie vom mehrlagigen Perzeptron genutzt wird. Weil Gauß-Verteilungen lokal sind, benötigen wir typischerweise viel mehr lokale Einheiten als für eine verteilte Repräsentation notwendig wären, besonders dann, wenn die Eingabe hochdimensional ist.

Im Fall des überwachten Lernens können wir dann diese neue lokale Repräsentation als Eingabe verwenden. Wenn wir ein Perzeptron einsetzen, erhalten wir

$$y^t = \sum_{h=1}^{H} w_h p_h^t + w_0 \,, \tag{12.14}$$

RADIALE BASISFUNKTION

wobei H die Anzahl an Basisfunktionen darstellt. Diese Struktur wird als Netz *radialer Basisfunktionen* (RBF) bezeichnet (Broomhead und Lowe 1988; Moody und Darken 1989). Normalerweise verwendet man keine RBF-Netze mit mehr als einer Schicht an Gaußschen Einheiten. H ist der Komplexitätsparameter, wie die Anzahl an verborgenen Einheiten in einem mehrlagigen Perzeptron. Bisher haben wir ihn mit k bezeichnet, wenn er der Anzahl an Zentren im Fall des unüberwachten Lernens entsprach.

Hier erkennen wir den Vorteil davon, p_h statt b_h zu verwenden. Da b_h gleich 0/1, würde Gleichung 12.14, wenn sie b_h statt p_h enthalten würde, eine stückweise konstante Approximation mit Diskontinuitäten an den

Grenzen der Regionen im Eingaberaum der Eingabeeinheiten liefern. Die Werte für p_h sind weich und führen zu einer geglätteten Approximation, wobei ein gewichteter Durchschnitt beim Übergang von einer Region zur nächsten errechnet wird. Es ist leicht zu sehen, dass so ein Netz insofern ein universeller Approximator ist, als dass es jede beliebige Funktion mit der gewünschten Genauigkeit approximieren kann, vorausgesetzt, es existieren genügend Einheiten. Wir können ein Raster im Eingaberaum für unsere gewünschte Genauigkeit erstellen, eine Einheit definieren, die für jedes Raster aktiv sein wird, und ihr ausgehendes Gewicht, w_h, auf den gewünschten Ausgabewert setzen.

Diese Architektur ähnelt stark den nichtparametrischen Schätzern, beispielsweise den Parzen-Fenstern, die wir in Kapitel 8 betrachtet haben, und p_h können als Kernfunktionen betrachtet werden. Der Unterschied ist der, dass wir jetzt keine Kernfunktion über alle Trainingsinstanzen haben, sondern sie mit Hilfe einer Clustermethode gruppieren, um mit weniger Kernen auszukommen. H, die Anzahl an Einheiten, ist der Komplexitätsparameter, mit dem Kompromisse zwischen der Einfachheit und der Genauigkeit geschlossen werden. Mit mehr Einheiten approximieren wir die Daten besser, erhalten jedoch ein komplexeres Modell und riskieren eine zu starke Anpassung; zu wenige Einheiten können zu einer zu geringen Anpassung führen. Erneut wird der optimale Wert mittels Kreuzvalidierung bestimmt.

Sobald $\boldsymbol{m}_h$ und s_h vorliegen und fest bestimmt sind, werden die p_h auch auf einen festen Wert gesetzt. Dann können die w_h leicht mit der Batch- oder Online-Methode trainiert werden. Im Falle der Regression handelt es sich um ein lineares Regressionsmodell (mit p_h als Eingaben) und die w_h können analytisch ohne jegliche Iteration gelöst werden (Abschnitt 4.6). Im Falle der Klassifikation müssen wir auf eine iterative Prozedur zurückgreifen. Wir haben Lernmethoden hierfür in Kapitel 10 besprochen und werden sie hier nicht noch einmal wiederholen.

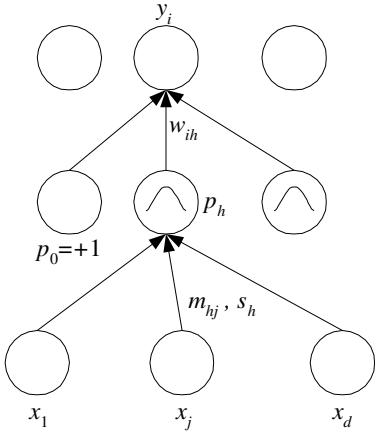


Abb. 12.7: Das RBF-Netz, bei dem p_h die verborgenen Einheiten sind, unter Nutzung der glockenförmigen Aktivierungsfunktion. $\boldsymbol{m}_h, s_h$ sind die Parameter der ersten Schicht und $\boldsymbol{w}_i$ sind die Gewichte der zweiten Schicht.

HYBRIDES LERNEN

Unser Vorgehen hier ist ein Zweistufenprozess: wir nutzen eine unüberwachte Methode zur Bestimmung der Zentren und verwenden darauf folgend eine überwachte Schicht. Hierbei spricht man vom *hybriden Lernen*. Wir können auch alle Parameter, inklusive $\boldsymbol{m}_h$ und s_h, auf überwachte Weise erlernen. Die radiale Basisfunktion von Gleichung 12.13 ist differenzierbar und wir können rückpropagieren, genauso wie wir bei einem mehrlagigen Perzeptron rückpropagierten, um die Gewichte der ersten Schicht zu aktualisieren. Die Struktur ähnelt einem mehrlagigen Perzeptron mit p_h als verborgenen Einheiten, $\boldsymbol{m}_h$ und s_h als Parameter der ersten Schicht, der Gauß-Funktion als Aktivierungsfunktion in der verborgenen Schicht und w_h als Gewichte der zweiten Schicht (siehe Abbildung 12.7).

ANKERMETHODE

Bevor wir das jedoch weiter vertiefen, soll noch einmal daran erinnert sein, dass das Training eines zweischichtigen Netzes eine langsame Angelegenheit ist. Beim hybriden Lernen wird eine Schicht nach der anderen trainiert und die Methode ist schneller. Eine andere Technik, die sogenannte *Ankermethode*, legt die Zentren auf zufällig gewählte Muster aus dem Trainingsdatensatz ohne weitere Aktualisierung. Wenn viele Einheiten existieren, ist dies eine adäquate Methode.

Andererseits ist die Genauigkeit normalerweise nicht so hoch wie bei Nutzung einer komplett überwachten Methode. Man betrachte den Fall, wenn die Eingabe gleichverteilt ist. Dann platziert die k-Means-Clustermethode die Einheiten einheitlich. Ändert sich die Funktion signifikant in einem kleinen Teil des Raumes, ist es ratsamer, möglichst viele Zentren an Stellen zu haben, an denen die Funktion sich schnell ändert, um den Fehler so niedrig wie möglich zu halten; das entspricht dem, was die komplett überwachte Methode tun würde.

Befassen wir uns nun damit, wie all diese Parameter auf eine komplett überwachte Art und Weise trainiert werden können. Der Ansatz ist der gleiche wie bei der auf mehrlagige Perzeptronen angewandten Rückpropagierung. Betrachten wir den Fall der Regression mit multiplen Ausgaben. Der Batch-Fehler ist

$$E(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \,, \tag{12.15}$$

wobei

$$y_i^t = \sum_{h=1}^{H} w_{ih} p_h^t + w_{i0} \,. \tag{12.16}$$

Unter Verwendung des Gradientenabstiegs erhalten wir die folgende Aktualisierungsregel für die Gewichte der zweiten Schicht:

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) p_h^t \,. \tag{12.17}$$

Hierbei handelt es sich um die übliche Aktualisierungsregel für Perzeptronen, mit p_h als Eingaben. Typischerweise überschneiden sich die p_h kaum, und bei jeder Iteration sind lediglich wenige p_h ungleich 0 und nur deren w_h werden aktualisiert. Deshalb lernen RBF-Netze sehr schnell und auch schneller als mehrlagige Perzeptronen, die eine verteilte Repräsentation nutzen.

Auf die gleiche Weise können wir die Aktualisierungsgleichungen für die Zentren und Streuungen durch die Rückpropagierung erhalten (Kettenregel):

$$\Delta m_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{(x_j^t - m_{hj})}{s_h^2} , \tag{12.18}$$

$$\Delta s_h = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) w_{ih} \right] p_h^t \frac{\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2}{s_h^3} . \tag{12.19}$$

Vergleichen wir einmal Gleichung 12.18 mit Gleichung 12.5. Erstens nutzen wir hier p_h statt b_h, was bedeutet, dass nicht nur die nächstliegende, sondern alle Einheiten aktualisiert werden, in Abhängigkeit von ihren Zentren und Streuungen. Zweitens wird hier die Aktualisierung überwacht und enthält den rückpropagierten Fehlerterm. Die Aktualisierung hängt nicht nur von der Eingabe ab, sondern auch vom finalen Fehler $(r_i^t - y_i^t)$, dem Effekt der Einheit auf die Ausgabe w_{ih}, der Aktivierung der Einheit p_h und der Eingabe $(\boldsymbol{x} - \boldsymbol{m}_i)$.

In der Praxis brauchen die Gleichungen 12.18 und 12.19 etwas zusätzliche Kontrolle. Wir müssen explizit prüfen, dass die s_h weder sehr klein noch sehr groß und damit nutzlos werden; außerdem müssen wir prüfen, dass die $\boldsymbol{m}_h$ im gültigen Eingabebereich bleiben.

Im Falle der Klassifikation erhalten wir

$$y_i^t = \frac{\exp\left[\sum_h w_{ih} p_h^t + w_{i0}\right]}{\sum_k \exp\left[\sum_h w_{kh} p_h^t + w_{k0}\right]} \tag{12.20}$$

und der Kreuzentropiefehler ist

$$E(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h} | \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t . \tag{12.21}$$

Aktualisierungsregeln können auf ähnliche Weise unter Verwendung des Gradientenabstiegs abgeleitet werden (Übung 3).

Betrachten wir noch einmal Gleichung 12.14. Wenn p_h für eine beliebige Eingabe ungleich 0 ist, so ist der Beitrag zur Ausgabe w_h. Der Beitrag ist eine konstante Anpassung, wie sie durch w_h gegeben ist. Normalerweise überlappen sich Gauß-Verteilungen kaum und eine oder zwei von ihnen

haben einen Wert für p_h ungleich 0. In jedem Fall tragen nur wenige Einheiten etwas zur Ausgabe bei. w_0 ist die konstante Verschiebung und wird zur gewichteten Summe der aktiven Einheiten (ungleich Null) addiert. Wir sehen außerdem, dass $y = w_0$, falls alle p_h gleich 0 sind. Wir können deshalb w_0 als den „Standardwert“ für y betrachten: ist keine Gauß-Verteilung aktiv, so ist die Ausgabe durch diesen Wert gegeben. Eine Möglichkeit ist also die, dieses „Standardmodell“ mächtiger zu gestalten. Beispielsweise können wir annehmen:

$$y^t = \sum_{h=1}^{H} w_h p_h^t + \boldsymbol{v}^T \boldsymbol{x}^t + v_0 \; . \tag{12.22}$$

In diesem Fall ist das Standardmodell linear: $\boldsymbol{v}^T \boldsymbol{x}^t + v_0$. Wenn sie ungleich 0 sind, arbeiten Gauß-Verteilungen als „Ausnahmen“ und modifizieren die Ausgabe, um den Unterschied zwischen der gewünschten Ausgabe und der Ausgabe des Standardmodells zu kompensieren. Solch ein Modell kann auf überwachte Weise trainiert werden, und das Standardmodell kann zusammen mit w_h trainiert werden (Übung 4). In Abschnitt 17.11 diskutieren wir ein ähnliches Modell, die Kaskadierung, die wir als eine Kombination aus zwei Lernern auffassen, wobei der eine dieser Lerner eine allgemeine Regel ist, während der andere durch eine Menge von Ausnahmen gebildet wird.

12.4 Einbindung von regelbasiertem Wissen

A-PRIORI-WISSEN

Das Training eines beliebigen Lernsystems wird viel einfacher, wenn es uns möglich ist, *a priori bekanntes Wissen* einzuarbeiten, um das System zu initialisieren. Beispielsweise kann vorher bekanntes Wissen in der Form einer Menge an Regeln vorliegen, welche die Eingabe-/Ausgabeabbildung spezifizieren, die das Modell, zum Beispiel das RBF-Netz, zu erlernen hat. Dies ist häufig bei industriellen und medizinischen Anwendungen der Fall, bei denen Regeln durch Experten bereitgestellt werden können. Ähnlich dazu können Regeln, sobald ein Netz trainiert worden ist, aus der Lösung so extrahiert werden, dass die Lösung für das Problem besser zu verstehen ist.

Die Möglichkeit auf derlei vorher bekanntes Wissen zurückgreifen zu können, stellt einen zusätzlichen Vorteil dar, wenn das Netz in Regionen des Eingaberaums extrapolieren soll, aus denen es noch keine Trainingsdaten bekommen hat. Des Weiteren wird in vielen Steuerungsapplikationen vom Netz verlangt, dass es direkt von Beginn an vernünftige Entscheidungen trifft. Bis es genügend Trainingsdaten gesehen hat, muss das System sich daher primär auf dieses vorher bekannte Wissen verlassen.

In vielen Anwendungen liegen uns typischerweise einige Grundregeln vor, denen wir zu Beginn versuchen zu folgen, die aber dann aufgrund von Erfahrungen verfeinert und verändert werden. Je besser unser Ausgangswissen zu einem Problem, desto schneller können wir eine gute Leistungsfähigkeit erzielen und desto weniger Training ist von Nöten.

So eine Einbindung von vorher bekanntem Wissen oder die Extraktion von erlerntem Wissen ist leicht mit RBF-Netzen durchzuführen, da die Einheiten lokal sind. Dadurch vereinfacht sich die *Regelextraktion* (Tresp, Hollatz und Ahmad 1997). Ein Beispiel ist

Regelextraktion

$$\mathsf{IF}\ ((x_1 \approx a)\ \mathsf{AND}\ (x_2 \approx b))\ \mathsf{OR}\ (x_3 \approx c)\ \mathsf{THEN}\ y = 0{,}1\ , \quad (12.23)$$

wobei $x_1 \approx a$ bedeutet, „x_1 ist ungefähr gleich a“. Im Rahmen von RBF wird diese Regel durch zwei Gaußsche Einheiten codiert als

$$p_1 = \exp\left[-\frac{(x_1-a)^2}{2s_1^2}\right] \cdot \exp\left[-\frac{(x_2-b)^2}{2s_2^2}\right] \text{ mit } w_1 = 0{,}1\ ,$$
$$p_2 = \exp\left[-\frac{(x_3-c)^2}{2s_3^2}\right] \text{ mit } w_2 = 0{,}1\ .$$

„Ungefähr gleich“ wird durch eine Gauß-Verteilung modelliert, wobei das Zentrum dem Idealwert entspricht und die Streuung die erlaubte Abweichung um diesen Idealwert herum darstellt. Die Konjunktion ist das Produkt zweier univariater Gauß-Verteilungen, das heißt eine bivariate Gauß-Verteilung. Dann kann der erste Term des Produktes durch eine zweidimensionale Gauß-Verteilung, spricht mit $\boldsymbol{x} = [x_1, x_2]$, abgefertigt werden, die ihr Zentrum bei (a, b) hat, und die Streuungen an den zwei Dimensionen sind durch s_1 und s_2 gegeben. Die Disjunktion wird durch zwei separate Gauß-Verteilungen modelliert, wobei jede eine der Disjunkten bearbeitet.

Bei gegebenen und eingeordneten Trainingsdaten können die Parameter des so konstruierten RBF-Netzes nach der anfänglichen Konstruktion feinabgestimmt werden, wobei ein kleiner Wert für η verwendet wird.

Diese Formulierung ist mit dem Ansatz der Fuzzy-Logik verwandt, bei dem Gleichung 12.23 als *Fuzzy-Regel* bezeichnet wird. Die Gaußsche Basisfunktion, die auf ungefähre Gleichheit testet, entspricht einer *Fuzzy-Membership-Funktion* (Berthold 1999; Cherkassky und Mulier 1998).

Fuzzy-Regel

Fuzzy-Membership-Funktion

12.5 Normalisierte Basisfunktionen

In Gleichung 12.14 ist es für eine Eingabe möglich, dass alle der p_h gleich null sind. In einigen Anwendungen wünschen wir uns unter Umständen einen *Normalisierungsschritt*, um sicherzustellen, dass sich die Werte der

lokalen Einheiten zu eins summieren, und um somit zu garantieren, dass es mindestens eine Einheit ungleich null für jede beliebige Eingabe gibt:

$$g_h^t = \frac{p_h^t}{\sum_{l=1}^{H} p_l^t} = \frac{\exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2/2s_h^2]}{\sum_l \exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_l\|^2/2s_l^2]} \,. \tag{12.24}$$

Ein Beispiel ist in Abbildung 12.8 dargestellt. Nehmen wir p_h als $p(\boldsymbol{x}|h)$, so entsprechen die g_h der a-posteriori-Wahrscheinlichkeit dafür, dass $\boldsymbol{x}$ zu Einheit h gehört, $p(h|\boldsymbol{x})$. Es ist, als ob die Einheiten den Eingaberaum untereinander aufteilen. Wir können uns g_h selbst als Klassifikator vorstellen, der die verantwortliche Einheit für eine gegebene Eingabe wählt. Diese Klassifikation wird basierend auf Entfernungen durchgeführt, wie bei einem parametrischen Gaußschen Klassifikator (Kapitel 5).

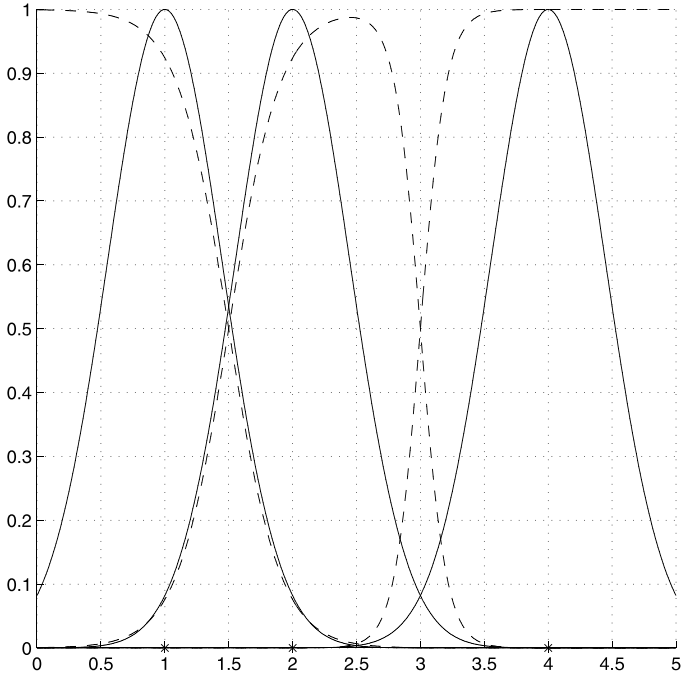


***Abb. 12.8*:** Vor (-) und nach (- -) der Normalisierung für drei Gauß-Verteilungen, deren Zentren mit „*“ markiert sind. Man beachte, wie die Region ungleich 0 einer Einheit auch von den Positionen anderer Einheiten abhängt. Sind die Streuungen klein, so implementiert die Normalisierung eine schärfere Aufteilung; mit größeren Streuungen überschneiden sich die Einheiten mehr.

Die Ausgabe ist eine gewichtete Summe

$$y_i^t = \sum_{h=1}^{H} w_{ih} g_h^t \,, \tag{12.25}$$

wobei es keinen Bedarf für einen Verzerrungsterm gibt, da es mindestens ein g_h ungleich 0 für jedes $\boldsymbol{x}$ gibt. Nutzen wir g_h statt p_h, so werden keinerlei zusätzliche Parameter eingeführt; die Einheiten werden lediglich zusammengekuppelt. p_h hängt nur von $\boldsymbol{m}_h$ und s_h ab, jedoch ist g_h aufgrund der Normalisierung von den Zentren und Streuungen aller Einheiten abhängig.

Im Falle der Regression erhalten wir die folgenden Aktualisierungsregeln unter Verwendung des Gradientenabstiegs:

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) g_h^t \,, \tag{12.26}$$

$$\Delta m_{hj} = \eta \sum_t \sum_i (r_i^t - y_i^t)(w_{ih} - y_i^t) g_h^t \frac{(x_j^t - m_{hj})}{s_h^2} \,. \tag{12.27}$$

Die Aktualisierungsregel für s_h sowie die Regeln für die Klassifikation können auf ähnliche Weise abgeleitet werden. Vergleichen wir diese einmal mit den Aktualisierungsregeln für das RBF-Netz mit nicht normalisierten Gauß-Verteilungen (Gleichung 12.17). Hier nutzen wir g_h statt p_h, wodurch die Aktualisierung einer Einheit nicht nur von ihren eigenen Parametern, sondern auch von den Zentren und Streuungen anderer Einheiten abhängig gemacht wird. Beim Vergleich von Gleichung 12.27 mit Gleichung 12.18 sehen wir, dass wir $(w_{ih} - y_i^t)$ statt w_{ih} haben, was die Rolle der Normalisierung für die Ausgabe verdeutlicht. Die „verantwortliche“ Einheit versucht, die Differenz zwischen ihrer Ausgabe, w_{ih}, und der finalen Ausgabe, y_i^t, zu verringern, und zwar proportional zu ihrer Verantwortlichkeit, g_h.

12.6 Kompetitive Basisfunktionen

Wie wir bisher gesehen haben, wird in einem RBF-Netz die finale Ausgabe als eine gewichtete Summe der Beiträge der lokalen Einheiten bestimmt. Zwar sind die Einheiten lokal, jedoch ist es die finale gewichtete Summe, die von Bedeutung ist und die wir so nah wie möglich an die verlangte Ausgabe heranführen möchten. In der Regression beispielsweise minimieren wir Gleichung 12.15, welche auf dem probabilistischen Modell basiert,

$$p(\boldsymbol{r}^t | \boldsymbol{x}^t) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(r_i^t - y_i^t)^2}{2\sigma^2} \right] \,, \tag{12.28}$$

wobei y_i^t durch Gleichung 12.16 (nicht normalisiert) oder Gleichung 12.25 (normalisiert) gegeben ist. In jedem Fall können wir das Modell als ein *kooperatives* ansehen, da die Einheiten kooperieren, um die finale Ausgabe y_i^t zu generieren. Nun diskutieren wir den Ansatz, bei dem *kompetitive Basisfunktionen* verwendet werden, wobei wir davon ausgehen, dass die Ausgabe aus einem Mischmodell gezogen wird:

KOMPETITIVE BASISFUNKTIONEN

$$p(\boldsymbol{r}^t|\boldsymbol{x}^t) = \sum_{h=1}^{H} p(h|\boldsymbol{x}^t)p(\boldsymbol{r}^t|h,\boldsymbol{x}^t)\,. \tag{12.29}$$

$p(h|\boldsymbol{x}^t)$ sind die Anteile der Mischungskomponenten und $p(\boldsymbol{r}^t|h,\boldsymbol{x}^t)$ ist die Wahrscheinlichkeit dafür, dass die Mischungskomponente h gegeben die Eingabe $\boldsymbol{x}^t$ die gewünschte Ausgabe $\boldsymbol{r}^t$ produziert hat. Man beachte, dass beide dieser Terme von der Eingabe $\boldsymbol{x}$ abhängen.

Die Anteile der Mischungskomponenten sind

$$p(h|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|h)p(h)}{\sum_l p(\boldsymbol{x}|l)p(l)}\,, \tag{12.30}$$

$$g_h^t = \frac{a_h \exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2/2s_h^2]}{\sum_l a_l \exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_l\|^2/2s_l^2]}\,. \tag{12.31}$$

Wir gehen im Allgemeinen davon aus, dass die a_h gleich sind und ignorieren sie. Befassen wir uns zuerst mit dem Fall der Regression mit ihren Gaußkomponenten. In Gleichung 12.28 wird Rauschen zur gewichteten Summe hinzugefügt; hier wird eine Komponente gewählt und Rauschen wird ihrer Ausgabe y_{ih}^t zugefügt.

Unter Verwendung des Mischmodells aus Gleichung 12.29 ergibt sich die Log-Likelihood als

$$\mathcal{L}(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \exp\left[-\frac{1}{2}\sum_i (r_i^t - y_{ih}^t)^2\right], \tag{12.32}$$

wobei $y_{ih}^t = w_{ih}$ die durch Komponente h vorgenommene konstante Anpassung für die Ausgabe i repräsentiert, welche streng genommen nicht von $\boldsymbol{x}$ abhängt. (In Abschnitt 12.8.2 behandeln wir den Fall einer kompetitiven Mischung aus Expertensystemen, bei der die lokale Anpassung eine lineare Funktion von $\boldsymbol{x}$ ist.) Wie wir erkennen, liegt im Falle von g_h^t gleich 1 die Verantwortung für die Generierung der richtigen Ausgabe bei dieser Komponente und sie muss den quadrierten Fehler ihrer Vorhersage, $\sum_i (r_i^t - y_{ih}^t)^2$, minimieren.

Unter Verwendung des Gradientenanstiegs, um die Log-Likelihood zu maximieren, erhalten wir

$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_{ih}^t) f_h^t \tag{12.33}$$

mit

$$f_h^t = \frac{g_h^t \exp[-\frac{1}{2}\sum_i (r_i^t - y_{ih}^t)^2]}{\sum_l g_l^t \exp[-\frac{1}{2}\sum_i (r_i^t - y_{il}^t)^2]}\,, \tag{12.34}$$

$$p(h|\boldsymbol{r}, \boldsymbol{x}) = \frac{p(h|\boldsymbol{x}) p(\boldsymbol{r}|h, \boldsymbol{x})}{\sum_l p(l|\boldsymbol{x}) p(\boldsymbol{r}|l, \boldsymbol{x})}\,. \tag{12.35}$$

$g_h^t \equiv p(h|\boldsymbol{x}^t)$ ist die a-posteriori-Wahrscheinlichkeit von Einheit h bei vorliegender Eingabe und hängt von den Zentren und Streuungen aller Einheiten ab. $f_h^t \equiv p(h|\boldsymbol{r}, \boldsymbol{x}^t)$ ist die a-posteriori-Wahrscheinlichkeit der Einheit h bei gegebener Eingabe und gewünschter Ausgabe, wobei auch der Fehler bei der Auswahl der verantwortlichen Einheit beachtet wird.

Auf ähnliche Weise können wir eine Regel ableiten, um die Zentren zu aktualisieren:

$$\Delta m_{hj} = \eta \sum_t (f_h^t - g_h^t) \frac{(x_j^t - m_{hj})}{s_h^2}\,. \tag{12.36}$$

f_h ist die a-posteriori-Wahrscheinlichkeit von Einheit h, die ebenfalls die verlangte Ausgabe in die Betrachtung mit einbezieht, wohingegen g_h die a-posteriori-Wahrscheinlichkeit ist, die nur die Eingabe beachtet. Ihre Differenz ist der Fehlerterm für die Zentren. Δs_h kann ähnlich abgeleitet werden. Im kooperativen Fall wirkt keinerlei Kraft auf die zu lokalisierenden Einheiten. Um den Fehler zu verringern, können die Mittelwerte und Streuungen beliebige Werte annehmen; es ist manchmal sogar möglich, dass die Streuungen sich vergrößern und noch weiter ausdehnen. Um beim kompetitiven Fall jedoch die Likelihood zu erhöhen, werden Einheiten zur Lokalisierung mit stärkerer Separation zwischen ihnen und kleineren Streuungen gezwungen.

Bei der Klassifikation ist jede Komponente für sich eine (Multinomial-)verteilung. Dann ergibt sich die Log-Likelihood als

$$\mathcal{L}(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \sum_t \log \sum_h g_h^t \prod_i (y_{ih}^t)^{r_i^t} \tag{12.37}$$

$$= \sum_t \log \sum_h g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right] \tag{12.38}$$

mit

$$y_{ih}^t = \frac{\exp w_{ih}}{\sum_k \exp w_{kh}}\,. \tag{12.39}$$

Aktualisierungsregeln für $w_{ih}, \boldsymbol{m}_h$ und s_h können mit Hilfe des Gradientenanstiegs abgeleitet werden, in welchem Folgendes beinhaltet ist:

$$f_h^t = \frac{g_h^t \exp[\sum_i r_i^t \log y_{ih}^t]}{\sum_l g_l^t \exp[\sum_i r_i^t \log y_{il}^t]}\,. \tag{12.40}$$

In Kapitel 7 haben wir den EM-Algorithmus für die Anpassung von gemischten Gauß-Verteilungen an Daten diskutiert. Es ist möglich, den EM auch für das überwachte Lernen zu generalisieren. Um genau zu sein, entspricht die Berechnung von f_h^t dem E-Schritt. $f_h^t \equiv p(\boldsymbol{r}|h, \boldsymbol{x}^t)$ ersetzt $p(h|\boldsymbol{x}^t)$, was wir im E-Schritt in Kapitel 7 verwendet haben, als es sich um unüberwachte Applikationen handelte. Im M-Schritt für die Regression aktualisieren wir die Parameter als

$$\boldsymbol{m}_h = \frac{\sum_t f_h^t \boldsymbol{x}^t}{\sum_t f_h^t}, \tag{12.41}$$

$$\mathbf{S}_h = \frac{\sum_t f_h^t (\boldsymbol{x}^t - \boldsymbol{m}_h)(\boldsymbol{x}^t - \boldsymbol{m}_h)^T}{\sum_t f_h^t}, \tag{12.42}$$

$$w_{ih} = \frac{\sum_t f_h^t r_i^t}{\sum_t f_h^t}. \tag{12.43}$$

Wir erkennen, dass w_{ih} ein gewichteter Durchschnitt ist, bei dem die Gewichte bei vorliegender Eingabe und gewünschter Ausgabe den a-posteriori-Wahrscheinlichkeiten der Einheiten entsprechen. Im Falle der Klassifikation gibt es für den M-Schritt keine analytische Lösung und wir müssen auf eine iterative Prozedur zurückgreifen, beispielsweise auf den Gradientenanstieg (Jordan und Jacobs 1994).

12.7 Lernen mit Vektorquantisierung

Angenommen, wir haben H Einheiten für jede Klasse, die bereits jenen Klassen zugeordnet sind. Diese Einheiten werden mit zufälligen Instanzen aus ihren Klassen initialisiert. Bei jeder Iteration finden wir die Einheit $\boldsymbol{m}_i$, die in der Euklidischen Distanz der Eingabeinstanz am nächsten liegt und nutzen die folgende Aktualisierungsregel:

$$\left.\begin{array}{ll} \Delta\boldsymbol{m}_i = \eta(\boldsymbol{x}^t - \boldsymbol{m}_i) & \text{falls } \boldsymbol{x}^t \text{ und } \boldsymbol{m}_i \text{ derselben} \\ & \text{Klasse zugeordnet sind,} \\ \Delta\boldsymbol{m}_i = -\eta(\boldsymbol{x}^t - \boldsymbol{m}_i) & \text{andernfalls.} \end{array}\right\} \tag{12.44}$$

Wenn das nächstliegende Zentrum die korrekte Klassenzuordnung besitzt, wird es in Richtung der Eingabe verschoben, um sie besser zu repräsentieren. Gehört es zur falschen Klasse, so wird es von der Eingabe weg bewegt, in der Erwartung, dass bei einer ausreichenden Verschiebung in einer zukünftigen Iteration ein Zentrum der korrekten Klasse das nächstliegende sein wird. Hierbei handelt es sich um das Modell des *Lernens mit Vektorquantisierung* (LVQ), welches von Kohonen (1990, 1995) entworfen wurde.

Lernen mit Vektorquantisierung

Die LVQ-Aktualisierungsgleichung ist analog zu Gleichung 12.36, in der die Richtung, in die das Zentrum verschoben wird, von der Differenz

zwischen zwei Werten abhängt: von unserer Vorhersage der Gewinnereinheit basierend auf den Eingabeinstanzen und von der Einheit, welche basierend auf der verlangten Ausgabe tatsächlich gewinnen sollte.

12.8 Gemischte Expertensysteme

In RBF-Netzen stellen wir entsprechend eines jeden lokalen Teilstücks eine konstante Anpassung zur Verfügung. In dem Fall, dass für eine beliebige Eingabe ein g_h mit dem Wert 1 und alle anderen mit dem Wert 0 vorliegen, erhalten wir eine stückweise konstante Approximation, bei der für die Ausgabe i die lokale Anpassung durch Teilstück h durch w_{ih} gegeben ist. Von der Taylorschen Erweiterung wissen wir, dass an jedem Punkt die Funktion geschrieben werden kann als:

$$f(x) = f(a) + (x - a)f'(a) + \cdots . \tag{12.45}$$

Somit ist eine konstante Approximation gut, wenn x nahe genug an a liegt und $f'(a)$ nahe 0 ist, das heißt, wenn $f(x)$ um a herum flach ist. Ist dies nicht der Fall, so müssen wir den Raum in eine große Zahl an Teilstücken zerlegen, was aufgrund des Fluchs der Dimensionalität bei hoher Eingabedimensionalität besonders kritisch ist.

STÜCKWEISE LINEARE APPROXIMATION GEMISCHTE EXPERTENSYSTEME

Eine Alternative bietet eine *stückweise lineare Approximation*, indem der nächste Term in der Taylorschen Erweiterung einbezogen wird, nämlich der lineare Term. Genau das wird durch *gemischte Expertensysteme* vollzogen (Jacobs et al. 1991). Wir schreiben

$$y_i^t = \sum_{h=1}^{H} w_{ih} g_h^t , \tag{12.46}$$

was Gleichung 12.25 entspricht, nur dass hier w_{ih}, der Beitrag von Teilstück h zur Ausgabe i, keine konstante, sondern eine lineare Funktion der Eingabe ist:

$$w_{ih}^t = \boldsymbol{v}_{ih}^T \boldsymbol{x}^t . \tag{12.47}$$

$\boldsymbol{v}_{ih}$ ist der Parametervektor, welcher die lineare Funktion definiert und einen Verzerrungsterm enthält, wodurch aus den gemischten Expertensystemen eine Generalisierung des RBF-Netzes wird. Eine Einheitsaktivierung kann als normalisierte RBF betrachtet werden:

$$g_h^t = \frac{\exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_h\|^2 / 2s_h^2]}{\sum_l \exp[-\|\boldsymbol{x}^t - \boldsymbol{m}_l\|^2 / 2s_l^2]} . \tag{12.48}$$

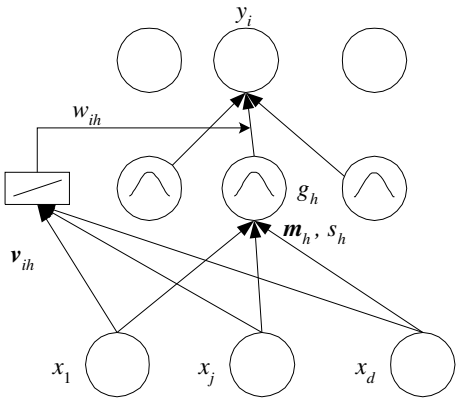


Abb. 12.9: Die gemischten Expertensysteme können als ein RBF-Netz gesehen werden, bei dem die Gewichte der zweiten Schicht Ausgaben von linearen Modellen sind. Aus Gründen der Einfachheit ist nur ein lineares Modell dargestellt.

Das Ganze kann man als RBF-Netz sehen, abgesehen davon, dass die Gewichte der zweiten Schicht keine Konstanten sind, sondern Ausgaben von linearen Modellen (siehe Abbildung 12.9). Jacobs et al. (1991) betrachten das aus einer anderen Perspektive: sie sehen $\boldsymbol{w}_h$ als lineare Modelle, wobei jedes die Eingabe aufnimmt, und bezeichnen sie als *Expertensysteme*. g_h werden als die Ausgaben eines *Gatternetzes* (engl. *gating network*) angesehen. Das Gatternetz arbeitet wie ein Klassifikator, wobei die Ausgaben sich zu 1 summieren und die Eingabe einem der Expertensysteme zugewiesen wird (siehe Abbildung 12.10).

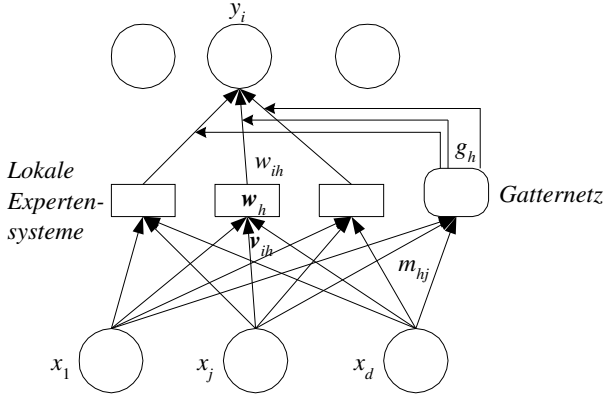


Abb. 12.10: Die gemischten Expertensysteme können als Modell betrachtet werden, das multiple Modelle kombiniert. $\boldsymbol{w}_h$ sind die Modelle und das Gatternetz ist ein weiteres Modell, welches das Gewicht eines jeden Modells, wie es durch g_h gegeben ist, bestimmt. Wenn man es auf diese Art betrachtet, sind weder die Expertensysteme noch das Gattermodell der Einschränkung unterworfen, linear sein zu müssen.

Wenn wir das Gatternetz auf diese Weise betrachten, kann jeder beliebige Klassifikator für das Gatter verwendet werden. Ist $\boldsymbol{x}$ hochdimensional,

kann die Verwendung von lokalen Gauß-Verteilungen unter Umständen eine große Zahl an Expertensystemen bedingen und Jacobs et al. (1991) schlagen vor,

$$g_h^t = \frac{\exp[\boldsymbol{m}_h^T \boldsymbol{x}^t]}{\sum_l \exp[\boldsymbol{m}_l^T \boldsymbol{x}^t]} \tag{12.49}$$

zu nutzen, wobei es sich um einen linearen Klassifikator handelt. Man beachte, dass $\boldsymbol{m}_h$ nicht länger Zentren sind, sondern Hyperebenen und als solche Verzerrungswerte beinhalten. Dieses Gatternetz implementiert eine Klassifikation, bei der es die Eingaberegion, für die Expertensystem h verantwortlich ist, linear von den Expertisenregionen anderer Expertensysteme abgrenzt. Wie wir in Kapitel 17 noch einmal sehen werden, sind die gemischten Expertensysteme eine generelle Architektur für die Kombination von multiplen Modellen; die Expertensysteme und das Gatter können nichtlinear sein, also zum Beispiel mehrlagige Perzeptronen statt linearer Perzeptronen enthalten (Übung 6).

Eine den gemischten Expertensystemen ähnliche Architektur und eine gleitende Linienglättung (Abschnitt 8.8.3) wurde von Bottou und Vapnik entworfen (1992). In deren Ansatz wird zu Beginn kein Training vorgenommen. Wird eine Testinstanz vorgelegt, so wird eine Teilmenge der Daten nahe der Testinstanz aus dem Trainingsdatensatz gewählt (wie beim k-Nächste-Nachbarn-Klassifikator aber mit einem großen k). Dann wird ein einfaches Modell, beispielsweise ein linearer Klassifikator, mit diesen lokalen Daten trainiert. Schließlich wird die Vorhersage für diese Instanz getroffen und das Modell wird verworfen. Für die nächste Instanz wird ein neues Modell erstellt, und so weiter. Bei einer Anwendung zur Erkennung handschriftlicher Ziffern hat dieses Modell einen geringeren Fehler als das mehrlagige Perzeptron, der k-Nächste-Nachbarn-Algorithmus und Parzen-Fenster; der Nachteil besteht darin, dass das neue Modell aus dem Stehgreif für jede Testinstanz trainiert werden muss.

12.8.1 Kooperative Expertensysteme

Im kooperativen Fall ist y_i^t durch Gleichung 12.46 gegeben und wir möchten, dass der Wert so nah wie möglich an der geforderten Ausgabe r_i^t liegt. Bei der Regression ist die Fehlerfunktion

$$E(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i (r_i^t - y_i^t)^2 \,. \tag{12.50}$$

Unter Nutzung des Gradientenabstiegs werden die Gewichtsparameter der zweiten Schicht (des Expertensystems) aktualisiert als

$$\Delta \boldsymbol{v}_{ih} = \eta \sum_t (r_i^t - y_i^t) g_h^t \boldsymbol{x}^t \,. \tag{12.51}$$

Verglichen mit Gleichung 12.26 sehen wir, dass der einzige Unterschied darin besteht, dass diese neue Aktualisierung eine Funktion der Eingabe ist.

Nutzen wir ein Softmax-Gatter (Gleichung 12.49), so erhalten wir unter Verwendung des Gradientenabstiegs die folgende Aktualisierungsregel für die Hyperebenen:

$$\Delta m_{hj} = \eta \sum_t \sum_i (r_i^t - y_i^t)(w_{ih}^t - y_i^t) g_h^t x_j^t \,. \tag{12.52}$$

Wenn wir ein radiales Gatter verwenden (Gleichung 12.48), so unterscheidet sich lediglich der letzte Term, $\partial p_h / \partial m_{hj}$.

Bei der Klassifikation haben wir

$$y_i = \frac{\exp\left[\sum_h w_{ih} g_h^t\right]}{\sum_k \exp\left[\sum_h w_{kh} g_h^t\right]} \tag{12.53}$$

mit $w_{ih} = \boldsymbol{v}_{ih}^T \boldsymbol{x}$, und Aktualisierungsregeln können mit Hilfe des Gradientenabstiegs abgeleitet werden, um die Kreuzentropie zu minimieren (Übung 7).

12.8.2 Kompetitive Expertensysteme

Wie bei einer kompetitiven RBF haben wir

$$\mathcal{L}(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h} | \mathcal{X}) = \sum_t \log \sum_h g_h^t \exp\left[-\frac{1}{2} \sum_i (r_i^t - y_{ih}^t)^2\right] \tag{12.54}$$

mit $y_{ih}^t = w_{ih}^t = \boldsymbol{v}_{ih} \boldsymbol{x}^t$. Unter Verwendung des Gradientenanstiegs erhalten wir

$$\Delta \boldsymbol{v}_{ih} = \eta \sum_t (r_i^t - y_{ih}^t) f_h^t \boldsymbol{x}^t \,, \tag{12.55}$$

$$\Delta \boldsymbol{m}_h = \eta \sum_t (f_h^t - g_h^t) \boldsymbol{x}^t \tag{12.56}$$

unter Annahme eines Softmax-Gatters wie in Gleichung 12.49 gegeben.

Bei der Klassifikation haben wir

$$\mathcal{L}(\{\boldsymbol{m}_h, s_h, w_{ih}\}_{i,h} | \mathcal{X}) = \sum_t \log \sum_h g_h^t \prod_i (y_{ih}^t)^{r_i^t} \tag{12.57}$$

$$= \sum_t \log \sum_h g_h^t \exp\left[\sum_i r_i^t \log y_{ih}^t\right] \tag{12.58}$$

mit

$$y_{ih}^t = \frac{\exp w_{ih}^t}{\sum_k \exp w_{kh}^t} = \frac{\exp[\boldsymbol{v}_{ih}\boldsymbol{x}^t]}{\sum_k \exp[\boldsymbol{v}_{kh}\boldsymbol{x}^t]} \,. \tag{12.59}$$

Jordan und Jacobs (1994) generalisieren den EM-Algorithmus für den kompetitiven Fall mit lokalen linearen Modellen. Alpaydın und Jordan (1996) vergleichen kooperative und kompetitive Modelle für Klassifikationsaufgaben und kommen zu dem Schluss, dass das kooperative Modell im Allgemeinen genauer ist, die kompetitive Version jedoch schneller lernt. Das ist deshalb so, weil sich Modelle im kooperativen Fall mehr überschneiden und eine glattere Approximation implementieren; somit ist dieser Fall bei Regressionsproblemen vorzuziehen. Das kompetitive Modell ermöglicht eine schärfere Aufteilung; im Allgemeinen ist nur ein Expertensystem für eine Eingabe aktiv und somit geht das Lernen schneller.

12.9 Hierarchisch gemischte Expertensysteme

In Abbildung 12.10 sehen wir eine Menge an Expertensystemen und ein Gatternetz, welches eins der Expertensysteme als eine Funktion der Eingabe auswählt. Bei *hierarchisch gemischten Expertensystemen* ersetzen wir jedes Expertensystem auf rekursive Weise mit einem kompletten System an gemischten Expertensystemen (Jordan und Jacobs 1994). Ist eine Architektur einmal ausgewählt – sprich die Tiefe, die Expertensysteme und die Gattermodelle – so kann der gesamte Baum anhand einer Stichprobe mit zugeordneten Instanzen gelernt werden. Jordan und Jacobs (1994) leiten für eine solche Architektur Lernregeln sowohl für den Gradientenabstieg als auch für den EM-Algorithmus ab (siehe Übung 9).

HIERARCHISCH GEMISCHTE EXPERTENSYSTEME

Wir können diese Struktur auch als Entscheidungsbaum interpretieren (Kapitel 9) und seine Gatternetze als Entscheidungsknoten. Bei den Entscheidungsbäumen, die wir zuvor diskutiert haben, trifft ein Entscheidungsknoten eine harte Entscheidung und wählt einen der Zweige, d. h., wir nehmen dort nur einen Zweig von der Wurzel bis zu einem der Blätter. Hier haben wir es dagegen mit einem *weichen Entscheidungsbaum* zu tun, denn da uns das Gattermodell eine Wahrscheinlichkeit zurückgibt, nehmen wir alle Zweige, nur mit unterschiedlichen Wahrscheinlichkeiten. Wir traversieren alle Pfade zu allen Blättern und nehmen eine gewichtete Summe über alle Blattwerte, wobei die Gewichte gleich dem Produkt der Gatterwerte auf dem Pfad zum jeweiligen Blatt sind. Diese Mittelung hat den Vorteil, dass die Grenzen zwischen Blattregionen nicht mehr hart sind. Vielmehr gibt es Übergänge von einem Blatt zum andern, was die Antwort glättet (İrsoy, Yıldız und Alpaydın 2012).

12.10 Anmerkungen

Ein RBF-Netz kann als ein neuronales Netz angesehen werden, welches durch ein Netz aus einfachen Verarbeitungseinheiten implementiert ist. Es unterscheidet sich von einem mehrlagigen Perzeptron, als dass die erste und zweite Schicht unterschiedliche Funktionen implementieren. Omohundro (1987) befasst sich damit, wie lokale Modelle als neuronale Netze implementiert werden können und spricht ebenfalls hierarchische Datenstrukturen für die schnelle Lokalisierung von relevanten lokalen Einheiten an. Specht (1991) zeigt, wie Parzen-Fenster als neuronales Netz implementiert werden können.

Platt (1991) entwarf eine inkrementelle Version des RBF-Netzes, bei der neue Einheiten je nach Bedarf zugefügt werden. Fritzke (1995) hat ähnlich dazu eine anwachsende Version für eine SOM vorgeschlagen.

Lee (1991) vergleicht die k-Nächste-Nachbarn-Methode, mehrlagige Perzeptronen und RBF-Netze anhand einer Applikation zur Erkennung handschriftlicher Ziffern und kommt zu dem Schluss, dass diese drei Methoden alle kleine Fehlerraten aufweisen. RBF-Netze lernen schneller als die Variante mit der Rückpropagierung an einem mehrlagigen Perzeptron, nutzen aber mehr Parameter. Diese beiden Methoden sind dem k-NN hinsichtlich der Klassifikationsgeschwindigkeit und des Speicherbedarfs überlegen. Für Anwendungen können eine kurze Rechenzeit, wenig Speicherbedarf oder geringe Komplexität der Berechnungen unter Umständen wichtiger sein als kleine Unterschiede in den Fehlerraten.

Kohonens SOM (1990, 1995) ist eine der populärsten Methoden neuronaler Netze, die bereits in einer Vielzahl von Applikationen Anwendung fand, inklusive der explorativen Datenanalyse und als Vorbearbeitungsstufe vor einem überwachten Lerner. Eine interessante und erfolgreiche Anwendung findet sich im Problem des Handlungsreisenden (Angeniol, Vaubois und Le Texier 1988). Ähnlich wie beim Unterschied zwischen der k-Means-Clustermethode und dem EM-Algorithmus für Gaußsche Mischungen (siehe Kapitel 7) ist das *generative topographische Mapping* (GTM) (Bishop, Svensén und Williams 1998) eine probabilistische Version von SOM. Sie optimiert die Log-Likelihood der Daten unter Verwendung einer Mischung von Gauß-Verteilungen, deren Mittelwerte auf einer zweidimensionalen Mannigfaltigkeit liegen müssen (für topologisches Ordnen in wenigen Dimensionen).

GENERATIVES TOPOGRAPHISCHES MAPPING

Nachdem in einem RBF-Netz die Mittelpunkte und Streuungen festgelegt wurden (beispielsweise durch die Wahl einer zufälligen Teilmenge von Trainingsinstanzen als Mittelpunkte wie bei der Ankermethode) ist das Trainieren der zweiten Schicht ein lineares Modell. Dieses Modell ist äquivalent zu den Support-Vektor-Maschinen mit Gauß-Kerneln, bei denen während des Lernens die beste Teilmenge von Instanzen – *Support-Vektoren* genannt – ausgewählt werden. Mit diesen Modellen beschäftigen

wir uns in Kapitel 13. Ähnlich sind auch Gaußsche Prozesse (Kapitel 16), mit denen wir auf der Basis gespeicherter Trainingsdaten interpolieren.

12.11 Übungen

1. Geben Sie ein RBF-Netz an, das die XOR-Funktion implementiert.

   Lösung: Es gibt zwei Möglichkeiten (siehe Abbildung 12.11). (a) Wir haben zwei kreisförmige Gauß-Verteilungen, die um die beiden positiven Instanzen zentriert sind, und die zweite Schicht verknüpft sie mit OR. (b) Wir haben eine elliptische Gauß-Verteilung, die um $(0{,}5, 0{,}5)$ mit negativer Korrelation zentriert ist, um die beiden positiven Instanzen abzudecken.

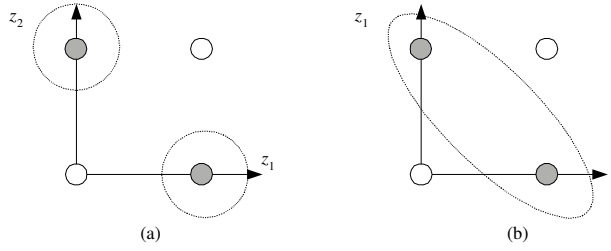


Abb. 12.11: Zwei Varianten zur Implementierung von XOR mit RBF.

2. Schreiben Sie ein RBF-Netz auf, das anstelle der radialen Einheiten wie in Gleichung 12.13 elliptische Einheiten verwendet.

   Lösung:

   $$p_h^t = \exp\left[-\frac{1}{2}(\boldsymbol{x}^t - \boldsymbol{m}_h)^T \mathbf{S}_h^{-1}(\boldsymbol{x}^t - \boldsymbol{m}_h) \right].$$

   Dabei ist $\mathbf{S}_h$ die lokale Kovarianzmatrix.

3. Leiten Sie die Aktualisierungsgleichungen für das RBF-Netz für die Klassifikation ab (Gleichung 12.20 und 12.21).

4. Zeigen Sie, wie das in Gleichung 12.22 gegebene System trainiert werden kann.

5. Vergleichen Sie die Anzahl an Parametern einer Architektur aus gemischten Expertensystemen mit einem RBF-Netz.

   Lösung: Mit d Eingaben, K Klassen und H Gauß-Verteilungen benötigt ein RBF-Netz $H \cdot d$ Parameter für die Mittelpunkte, H Parameter für die Streuungen und $(H+1)K$ Parameter für die

Gewichte der zweiten Schicht. Im Falle eines gemischten Expertensystems brauchen wir für jedes Gewicht der zweiten Schicht einen $(d+1)$-dimensionalen Vektor des linearen Modells, aber es gibt keine Verzerrung, so dass wir $HK(d+1)$ Parameter haben.

Beachten Sie, dass die Anzahl der Parameter in der ersten Schicht mit RBF die gleiche ist und dass sie unabhängig davon ist, ob wir ein Gaußsches oder ein Softmax-Gatter haben: Im Falle des Gaußschen Gatters brauchen wir für jede verborgene Einheit d Parameter für die Mittelpunkte und einen für die Streuung; beim Softmax-Gatter hat das lineare Modell $d+1$ Parameter (d Eingaben und eine Verzerrung).

6. Entwickeln Sie eine Architektur aus gemischten Expertensystemen, bei der die Expertensysteme und das Gatternetz mehrlagige Perzeptronen sind. Leiten Sie die Aktualisierungsgleichungen für die Regression und Klassifikation ab.

7. Leiten Sie die Aktualisierungsgleichungen für die kooperativen gemischten Expertensysteme für die Klassifikation ab.

8. Leiten Sie die Aktualisierungsgleichungen für die kompetitiven gemischten Expertensysteme für die Klassifikation ab.

9. Entwickeln Sie eine Architektur von hierarchisch gemischten Expertensystemen mit zwei Ebenen. Leiten Sie die Aktualisierungsgleichungen unter Verwendung des Gradientenabstiegs für die Regression und Klassifikation ab.

   Lösung: Die folgende Argumentation stammt von Jordan und Jacobs 1994; lediglich die Notation wurde ein wenig geändert, um sie mit der in diesem Buch verwendeten in Einklang zu bringen.

   Betrachten wir den Fall der Regression mit einer einzigen Ausgabe: y ist die Gesamtausgabe, die y_i sind die Ausgaben auf der ersten Ebene und die y_{ij} sind die Ausgaben auf der zweiten Ebene, die für ein Modell mit zwei Schichten aus den Blättern besteht. Entsprechend sind die g_i die Gatterausgaben auf der ersten Ebene und die $g_{j|i}$ sind die Ausgaben auf der zweiten Ebene, d. h. der Gatterwert des Experten j auf der zweiten Ebene, wenn wir auf der ersten Ebene den Zweig i gewählt haben:

$$y = \sum_i g_i y_i \,,$$

$$y_i = \sum_j g_{j|i} y_{ij} \quad \text{und} \quad g_i = \frac{\exp \boldsymbol{m}_i^T \boldsymbol{x}}{\sum_k \exp \boldsymbol{m}_k^T \boldsymbol{x}} \,,$$

$$y_{ij} = \boldsymbol{v}_{ij}^T \boldsymbol{x} \quad \text{und} \quad g_{j|i} = \frac{\exp \boldsymbol{m}_{ij}^T \boldsymbol{x}}{\sum_l \exp \boldsymbol{m}_{il}^T \boldsymbol{x}} \,.$$

Bei der Regression berechnet sich der zu minimierende Fehler wie folgt (wir verwenden hier eine kompetitive Version):

$$E = \sum_t \log \sum_i g_i^t \sum g_{j|i}^t \exp\left[-\frac{1}{2}(r^t - y_{ij}^t)^2 \right] .$$

Mittels Gradientenabstieg erhalten wir die folgenden Aktualisierungsgleichungen:

$$\Delta \boldsymbol{v}_{ij} = \eta \sum_t f_i^t f_{j|i}^t (r^t - y^t)\boldsymbol{x}^t ,$$

$$\Delta \boldsymbol{m}_i = \eta \sum_t (f_i^t - g_i^t)\boldsymbol{x}^t ,$$

$$\Delta \boldsymbol{m}_{ij} = \eta \sum_t f_i^t (f_{j|i}^t - g_{j|i}^t)\boldsymbol{x}^t ,$$

wobei wir die folgenden a-posteriori-Wahrscheinlichkeiten verwendet haben:

$$f_i^t = \frac{g_i^t \sum_j g_{j|i}^t \exp\left[-(1/2)(r^t - y_{ij}^t)^2 \right]}{\sum_k g_k^t \sum_j g_{j|k}^t \exp\left[-(1/2)(r^t - y_{kj}^t)^2 \right]} ,$$

$$f_{j|i}^t = \frac{g_{j|i}^t \exp\left[-(1/2)(r^t - y_{ij}^t)^2 \right]}{\sum_l g_{l|i}^t \exp\left[-(1/2)(r^t - y_{il}^t)^2 \right]} ,$$

$$f_{ij}^t = \frac{g_i^t g_{j|i}^t \exp\left[-(1/2)(r^t - y_{ij}^t)^2 \right]}{\sum_k g_k^t \sum_l g_{l|k}^t \exp\left[-(1/2)(r^t - y_{kl}^t)^2 \right]} .$$

Beachten Sie, wie wir die Gatterwerte entlang des Pfades, startend bei der Wurzel bis zu einem Blatt, multiplizieren.

Für eine Klassifikation mit $K > 2$ besteht eine Möglichkeit darin, wie zuvor K separate hierarchisch gemischte Expertensysteme zu verwenden (jeweils mit einzelnen Experten als Ausgabe), auf deren Ausgaben wir Softmax anwenden, um die Log-Likelihood zu maximieren:

$$\mathcal{L} = \sum_t \log \sum_i g_i^t \sum_j g_{j|i}^t \exp\left[\sum_c r_c^t \log p_c^t \right] ,$$

$$p_c^t = \frac{\exp y_c^t}{\sum_k \exp y_k^t} .$$

Dabei bezeichnet y_c^t die Ausgabe eines hierarchisch gemischten Expertensystems mit einer Ausgabe. Der interessantere Fall eines einzelnen hierarchisch gemischten Expertensystems mit mehreren Klassen, bei denen die Experten K Softmax-Ausgaben haben, wird in Waterhouse und Robinson 1994 diskutiert.

10. Bei gemischten Expertensystemen spezialisieren sich unterschiedliche Experten auf unterschiedliche Teile des Eingaberaums, weshalb es nötig sein kann, dass sie sich auf unterschiedliche Eingaben konzentrieren. Diskutieren Sie, wie die Dimensionalität in den Experten lokal reduziert werden kann.

12.12 Literaturangaben

Alpaydın, E., and M. I. Jordan. 1996. „Local Linear Perceptrons for Classification." *IEEE Transactions on Neural Networks* 7: 788–792.

Angeniol, B., G. Vaubois, and Y. Le Texier. 1988. „Self Organizing Feature Maps and the Travelling Salesman Problem." *Neural Networks* 1: 289–293.

Berthold, M. 1999. „Fuzzy Logic." In *Intelligent Data Analysis: An Introduction,* ed. M. Berthold and D. J. Hand, 269–298. Berlin: Springer.

Bishop, C. M., M. Svensén, and C. K. I. Williams. 1998. „GTM: The Generative Topographic Mapping." *Neural Computation* 10: 215–234.

Bottou, L., and V. Vapnik. 1992. „Local Learning Algorithms." *Neural Computation* 4: 888–900.

Broomhead, D. S., and D. Lowe. 1988. „Multivariable Functional Interpolation and Adaptive Networks." *Complex Systems* 2: 321–355.

Carpenter, G. A., and S. Grossberg. 1988. „The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network." *IEEE Computer* 21(3): 77–88.

Cherkassky, V., and F. Mulier. 1998. *Learning from Data: Concepts, Theory, and Methods.* New York: Wiley.

DeSieno, D. 1988. „Adding a Conscience Mechanism to Competitive Learning." In *IEEE International Conference on Neural Networks,* 117–124. Piscataway, NJ: IEEE Press.

Feldman, J. A., and D. H. Ballard. 1982. „Connectionist Models and their Properties." *Cognitive Science* 6: 205–254.

Fritzke, B. 1995. „Growing Cell Structures: A Self Organizing Network for Unsupervised and Supervised Training." *Neural Networks* 7: 1441–1460.

Grossberg, S. 1980. „How does the Brain Build a Cognitive Code?" *Psychological Review* 87: 1–51.

Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation.* Reading, MA: Addison Wesley.

İrsoy, O., O. T. Yıldız, and E. Alpaydın. 2012. „Soft Decision Trees." *In International Conference on Pattern Recognition,* 1819–1822. Piscataway, NJ: IEEE Press.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton. 1991. „Adaptive Mixtures of Local Experts." *Neural Computation* 3: 79–87.

Jordan, M. I., and R. A. Jacobs. 1994. „Hierarchical Mixtures of Experts and the EM Algorithm." *Neural Computation* 6: 181–214.

Kohonen, T. 1990. „The Self-Organizing Map." *Proceedings of the IEEE* 78: 1464–1480.

Kohonen, T. 1995. *Self-Organizing Maps.* Berlin: Springer.

Lee, Y. 1991. „Handwritten Digit Recognition Using k-Nearest Neighbor, Radial Basis Function, and Backpropagation Neural Networks." *Neural Computation* 3: 440–449.

Mao, J., and A. K. Jain. 1995. „Artificial Neural Networks for Feature Extraction and Multivariate Data Projection." *IEEE Transactions on Neural Networks* 6: 296–317.

Moody, J., and C. Darken. 1989. „Fast Learning in Networks of Locally-Tuned Processing Units." *Neural Computation* 1: 281–294.

Oja, E. 1982. „A Simplified Neuron Model as a Principal Component Analyzer." *Journal of Mathematical Biology* 15: 267–273.

Omohundro, S. M. 1987. „Efficient Algorithms with Neural Network Behavior." *Complex Systems* 1: 273–347.

Platt, J. 1991. „A Resource Allocating Network for Function Interpolation." *Neural Computation* 3: 213–225.

Specht, D. F. 1991. „A General Regression Neural Network." *IEEE Transactions on Neural Networks* 2: 568–576.

Tresp, V., J. Hollatz, and S. Ahmad. 1997. „Representing Probabilistic Rules with Networks of Gaussian Basis Functions." *Machine Learning* 27: 173–200.

Waterhouse, S. R., and A. J. Robinson. 1994. „Classification Using Hierarchical Mixtures of Experts." In *IEEE Workshop on Neural Networks for Signal Processing,* 177–186. Piscataway, NJ: IEEE Press.

13 Kernel-Maschinen

Kernel-Maschinen sind Methoden, die den Margin maximieren, wobei das Modell als Summe von Einflüssen einer Teilmenge der Trainingsinstanzen geschrieben werden kann. Diese Einflüsse sind durch anwendungsspezifische Ähnlichkeitskernel gegeben. Wir werden uns in diesem Kapitel mit „Kernel-Versionen" für die Klassifikation, die Regression, das Rangordnungsproblem, die Ausreißererkennung und die Dimensionalitätsreduktion beschäftigen und dabei auch darauf eingehen, wie die Kernel gewählt und verwendet werden.

13.1 Einführung

Wir wenden uns nun einem anderen Ansatz für die lineare Klassifikation und Regression zu. Es sollte uns nicht überraschen, dass es selbst für den einfachen Fall eines linearen Modells so viele verschiedene Verfahren gibt. Lernalgorithmen haben unterschiedliche induktive Verzerrungen, unterschiedliche Annahmen und unterschiedliche Zielfunktionen, was auf unterschiedliche lineare Modelle führt.

Das Modell, das wir in diesem Kapitel diskutieren wollen, wurde zuerst unter der Bezeichnung *Support-Vektor-Maschine* (auch *Stützvektormaschine*) eingeführt und später zu sogenannten *Kernel-Maschinen* verallgemeinert. In den letzten Jahren ist es aus einer Reihe von Gründen sehr populär geworden:

1. Es handelt sich um ein diskriminanzbasiertes Verfahren und beachtet das Prinzip von Vapnik, wonach man niemals im ersten Schritt ein komplexeres Problem lösen sollte, sondern immer zuerst das eigentliche Problem (Vapnik 1995). Wenn zum Beispiel bei der Klassifikation die Aufgabe darin besteht, die Diskriminanzfunktion zu lernen, ist es nicht nötig, die Klassendichten $p(\boldsymbol{x}|C_i)$ oder die genauen Werte der a-posteriori-Wahrscheinlichkeiten $P(C_i|\boldsymbol{x})$ zu schätzen. Ebensowenig brauchen wir bei der Ausreißererkennung eine Schätzung für die vollständige Dichte $p(\boldsymbol{x})$, sondern es genügt, die Trennlinie zu finden, welche die $\boldsymbol{x}$ mit kleinen Wahrscheinlichkeiten $p(\boldsymbol{x})$ separiert, genauer gesagt, die $\boldsymbol{x}$ mit $p(\boldsymbol{x}) < \theta$ für einen bestimmten Schwellwert $\theta \in (0, 1)$.

2. Nach dem Training kann der Parameter des linearen Modells, der Gewichtsvektor, in Form einer Teilmenge der Trainingsmenge aufgeschrieben werden – das sind die sogenannten *Support-Vektoren* (oder *Stützvektoren*). Bei der Klassifikation sind dies die Instanzen, die nahe an der Trennlinie liegen. Sind diese Instanzen bekannt, dann ist mit ihrer Hilfe eine Wissensextraktion möglich: Es sind die unsicheren oder fehlerhaften Fälle, die in der Umgebung der Grenzen zwischen den Klassen liegen. Ihre Anzahl liefert uns eine Schätzung für den Generalisierungsfehler, und wie sich zeigen wird, werden wir in der Lage sein, eine Kernel-Version zu formulieren, wenn es uns gelingt, den Modellparameter durch eine Menge von Instanzen auszudrücken.

3. Wie wir in Kürze sehen werden, wird die Ausgabe als Summe der Einflüsse von Support-Vektoren geschrieben, und diese sind durch sogenannte *Kernel-Funktionen* gegeben, also anwendungsspezifische Maße für die Ähnlichkeit zwischen Instanzen. Weiter vorn haben wir nichtlineare Basisfunktionen behandelt, die es uns erlauben, die Eingabe in einen anderen Raum abzubilden, in dem eine lineare (glatte) Lösung möglich ist. Kernel-Funktionen basieren auf der gleichen Idee.

4. Bei den meisten Lernalgorithmen werden Datenpunkte durch Vektoren dargestellt, und gewöhnlich wird dann entweder das Skalarprodukt (etwa bei mehrlagigen Perzeptronen) oder der Euklidische Abstand (etwa bei Netzen radialer Basisfunktionen) verwendet. Eine Kernel-Funktion erlaubt es uns weiter zu gehen. Wenn wir beispielsweise zwei Graphen G_1 und G_2 haben, dann entspricht $K(G_1, G_2)$ der Anzahl gemeinsamer Wege, die wir berechnen können, ohne G_1 und G_2 explizit als Vektoren darstellen zu müssen.

5. Kernelbasierte Algorithmen werden als konvexe Optimierungsprobleme formuliert, und für diese gibt es ein eindeutigees Optimum. Obwohl das Auffinden dieses Optimums noch immer einen iterativen Prozess erfordern kann, haben wir damit eine wesentlich einfachere Aufgabe vor uns als zum Beispiel bei mehrlagigen Perzeptronen. Natürlich bedeutet dies nicht, dass es nicht irgendwelche Hyperparameter für die Modellauswahl gäbe; wir müssen den Algorithmus in jedem Fall von Hand an die Daten anpassen.

Wir beginnen unsere Diskussion mit der Klassifikation, um die Erkenntnisse dann auf die Regression, das Rangordnungsproblem, die Ausreißererkennung und schließlich auf die Dimensionalitätsreduktion zu verallgemeinern. Dabei werden wir feststellen, dass wir in allen Fällen im Wesentlichen dasselbe quadratische Programmschema haben, um die Trennbarkeit oder den Margin von Instanzen unter einer Nebenbedingung an die Glattheit der Lösung zu maximieren. Durch das Lösen dieses Problems erhalten wir die Support-Vektoren. Die Kernel-Funktion definiert

den Raum entsprechend des ihr zugrundeliegenden Ähnlichkeitsbegriffs, und sie ist eine gute Kernel-Funktion, wenn wir in dem zugehörigen Raum eine bessere Trennung bekommen.

13.2 Die optimal trennende Hyperebene

Beginnen wir wieder mit zwei Klassen und nutzen wir die Label $-1/+1$ für diese zwei Klassen. Die Stichprobe ist $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$ mit $r^t = +1$, falls $\boldsymbol{x}^t \in \mathcal{C}_1$, und $r^t = -1$, falls $\boldsymbol{x}^t \in \mathcal{C}_2$. Wir möchten $\boldsymbol{w}$ und w_0 finden, so dass gilt

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}^t + w_0 &\geq +1 \quad \text{für} \quad r^t = +1\,, \\ \boldsymbol{w}^T\boldsymbol{x}^t + w_0 &\leq -1 \quad \text{für} \quad r^t = -1\,, \end{aligned}$$

was umformuliert werden kann zu

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq +1\,. \tag{13.1}$$

Man beachte, dass wir nicht einfach

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq 0$$

fordern.

Wir wollen nicht nur, dass die Instanzen auf der richtigen Seite der Hyperebene liegen, sondern zum Zwecke besserer Generalisierung sollen sie außerdem in gewisser Entfernung liegen. Den Abstand von der Trennebene zu den Instanzen, die ihr auf jeder Seite am nächsten liegen, bezeichnet man auch als *Margin*, und wir wollen ihn für die bestmögliche Verallgemeinerung maximieren.

MARGIN

Bereits in Abschnitt 2.1 haben wir uns im Zusammenhang mit der Anpassung eines Rechtecks mit dem Konzept des Margins beschäftigt, und wir hatten dort festgestellt, dass es besser ist, das Rechteck in der Mitte zwischen S und G zu wählen, um einen atmenden Raum zu bekommen. Sollte das Rauschen eine Testinstanz ein wenig verschieben, wird sie bei dieser Wahl immer noch auf der richtigen Seite der Trennlinie liegen.

Jetzt, da unsere Hypothesenklasse Linien sind, ist entsprechend die *optimal trennende Hyperebene* diejenige, welche den Margin maximiert.

OPTIMAL TRENNENDE HYPEREBENE

Wir erinnern uns, dass nach Abschnitt 10.3 die Entfernung von $\boldsymbol{x}^t$ zur Diskriminanzfunktion

$$\frac{|\boldsymbol{w}^T\boldsymbol{x}^t + w_0|}{\|\boldsymbol{w}\|}$$

entspricht, was im Falle von $r^t \in \{-1, +1\}$ auch als

$$\frac{r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0)}{\|\boldsymbol{w}\|}$$

geschrieben werden kann, wofür wir zumindest einen Wert ρ erzielen wollen:

$$\frac{r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0)}{\|\boldsymbol{w}\|} \geq \rho, \ \forall t \, . \tag{13.2}$$

Wir möchten ρ maximieren, jedoch gibt es eine unendliche Zahl an Lösungen, die wir durch Skalierung von $\boldsymbol{w}$ erhalten können; um eine eindeutige Lösung zu finden, setzen wir $\rho\|\boldsymbol{w}\| = 1$ und minimieren somit $\|\boldsymbol{w}\|$, um den Margin zu maximieren. Die Aufgabenstellung kann somit definiert werden (siehe Cortes und Vapnik 1995; Vapnik 1995) als

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{unter} \quad r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq +1, \ \forall t \, . \tag{13.3}$$

Hierbei handelt es sich um ein Standardproblem der quadratischen Optimierung, dessen Komplexität von d abhängt und welches direkt gelöst werden kann, um $\boldsymbol{w}$ und w_0 zu finden. Dann existieren auf beiden Seiten der Hyperebene Instanzen, die $1/\|\boldsymbol{w}\|$ weit von der Ebene entfernt liegen und der Gesamtabstand (Margin) ist $2/\|\boldsymbol{w}\|$.

In Abschnitt 10.2 haben wir gesehen, dass im Falle eines nicht linear trennbaren Problems ein Trick darin besteht, statt eine nichtlineare Funktion anzupassen, lieber das Problem unter Nutzung nichtlinearer Basisfunktionen in einen neuen Raum abzubilden. Im Allgemeinen trifft es zu, dass dieser neue Raum viel mehr Dimensionen besitzt als der Originalraum; in so einem Fall interessiert uns eine Methode, deren Komplexität nicht von der Eingabedimensionalität abhängt.

Wenn wir die optimale Trennebene finden, können wir das Optimierungsproblem in eine Form bringen, deren Komplexität von N abhängig ist, also der Anzahl an Trainingsinstanzen, und nicht von d. Ein weiterer Vorteil dieser neuen Formulierung besteht darin, dass sie uns gestattet, die Basisfunktionen in Form von Kernel-Funktionen umzuschreiben, wie wir in Abschnitt 13.5 sehen werden.

Um die neue Formulierung zu erhalten, schreiben wir zuerst Gleichung 13.3 als ein uneingeschränktes Problem unter Nutzung der Lagrange-Multiplikatoren α^t:

$$\begin{aligned} L_p &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{t=1}^{N} \alpha^t \left[r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0)\right] \\ &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_t \alpha^t r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) + \sum_t \alpha^t \, . \end{aligned} \tag{13.4}$$

Dies sollte hinsichtlich $\boldsymbol{w}, w_0$ minimiert und bezüglich $\alpha^t \geq 0$ maximiert werden. Der Sattelpunkt stellt die Lösung dar.

Es handelt sich um ein Problem einer konvexen quadratischen Optimierung, weil der Hauptterm sowie die linearen Bedingungen konvex sind. Somit können wir das duale Problem äquivalent lösen, indem wir uns die Karush-Kuhn-Tucker-Bedingungen zu Nutze machen. Der duale Aspekt besteht darin, L_p bezüglich α^t unter der Bedingung zu *maximieren*, dass der Gradient von L_p bezüglich $\boldsymbol{w}$ und w_0 gleich 0 ist sowie dass $\alpha^t \geq 0$:

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = 0 \quad \Rightarrow \quad \boldsymbol{w} = \sum_t \alpha^t r^t \boldsymbol{x}^t \tag{13.5}$$

$$\frac{\partial L_p}{\partial w_0} = 0 \quad \Rightarrow \quad \sum_t \alpha^t r^t = 0 \tag{13.6}$$

Wenn wir dies in Gleichung 13.4 einsetzen, erhalten wir die duale Form:

$$\begin{aligned} L_d &= \frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w}) - \boldsymbol{w}^T \sum_t \alpha^t r^t \boldsymbol{x}^t - w_0 \sum_t \alpha^t r^t + \sum_t \alpha^t \\ &= -\frac{1}{2}(\boldsymbol{w}^T\boldsymbol{w}) + \sum_t \alpha^t \\ &= -\frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (\boldsymbol{x}^t)^T \boldsymbol{x}^s + \sum_t \alpha^t \,. \end{aligned} \tag{13.7}$$

die wir nur bezüglich α^t maximieren, wobei die folgenden Nebenbedingungen zu erfüllen sind:

$$\sum_t \alpha^t r^t = 0 \quad \text{und} \quad \alpha^t \geq 0, \ \forall t \,.$$

Dies kann unter Verwendung quadratischer Optimierungsmethoden gelöst werden. Die Größe des dualen Problems hängt von N ab, also der Stichprobengröße, und nicht von d, der Eingabedimensionalität. Die obere Schranke für die Zeitkomplexität ist $\mathcal{O}(N^3)$ und die untere Schranke für die Speicherkomplexität ist $\mathcal{O}(N^2)$.

Sobald wir für α^t lösen, erkennen wir, dass deren Anzahl zwar N ist, die meisten jedoch mit $\alpha^t = 0$ verschwinden und nur für einen kleinen Prozentsatz $\alpha^t > 0$ gilt. Die Menge an $\boldsymbol{x}^t$, deren $\alpha^t > 0$, sind die sogenannten *Support-Vektoren* (manchmal auch als *Stützvektoren* bezeichnet), und wie wir in Gleichung 13.5 gesehen haben, wird $\boldsymbol{w}$ als die gewichtete Summe dieser Instanzen geschrieben, die als Support-Vektoren ausgewählt werden. Diese sind die $\boldsymbol{x}^t$, welche die Bedingung

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) = 1$$

erfüllen und noch im Margin liegen. Wir können diese Tatsache nutzen, um w_0 anhand von beliebigen Support-Vektoren zu berechnen mit

$$w_0 = r^t - \boldsymbol{w}^T\boldsymbol{x}^t \,. \tag{13.8}$$

Um numerische Stabilität zu gewährleisten, ist es ratsam, dies für alle Support-Vektoren durchzuführen und den Durchschnitt zu berechnen. Die dadurch gefundenen Diskriminanten nennt man *Support-Vektor-Maschinen* (SVM) (siehe Abbildung 13.1).

SUPPORT-VEKTOR-MASCHINE

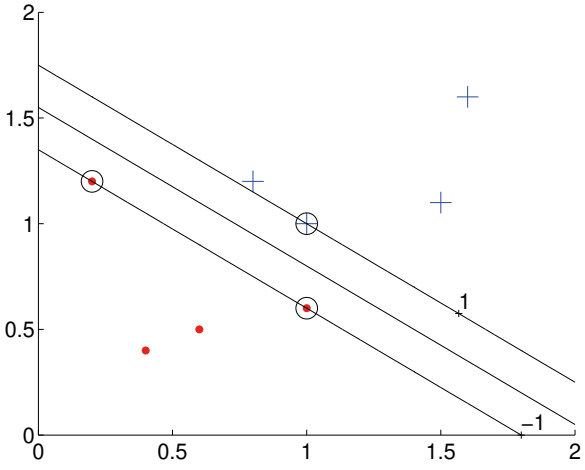


***Abb. 13.1*:** Ein Zwei-Klassen-Problem: Die Instanzen der Klassen sind durch Pluszeichen und Punkte symbolisiert, die durchgezogene Linie ist die Grenze und die Strichlinien definieren die Margins. Die mit Kreisen markieren Instanzen sind die Support-Vektoren.

Die Mehrheit der α^t sind gleich 0; für sie gilt $r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) > 1$. Dies sind die $\boldsymbol{x}^t$, die weiter innerhalb der Klassenregionen vom Margin aus liegen und und keinerlei Auswirkung auf die Hyperebene haben. Aus dieser Perspektive kann der Algorithmus mit dem verdichteten Nächste-Nachbarn-Algorithmus (Abschnitt 8.5) verglichen werden, welcher nur die Instanzen speichert, welche die Klassendiskriminante definieren.

Da es sich um eine diskriminanzbasierte Methode handelt, betrachtet die SVM nur die Instanzen, die nahe der Grenze liegen und verwirft jene, die weiter innerhalb der Klassengebiete zu finden sind. Unter Nutzung dieses Konzepts ist es möglich, einen einfacheren Klassifikator vor der SVM anzuwenden, um eine große Menge solcher Instanzen herauszufiltern und dadurch die Komplexität des Optimierungsschrittes der SVM zu verringern (Übung 1).

Während der Testphase bestehen wir nicht zwingend auf einem bestimmten Margin. Wir berechnen $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$ und wählen anhand des Vorzeichens von $g(\boldsymbol{x})$ aus:

$$\text{Wähle } C_1, \text{ falls } g(\boldsymbol{x}) > 0 \text{ und andernfalls } C_2\,.$$

13.3 Der nicht trennbare Fall: Soft-Margin-Trennebenen

Wenn die Daten nicht linear trennbar sind, ist der bereits besprochene Algorithmus nicht anwendbar. In so einem Fall, wenn zwei Klassen nicht linear getrennt werden können, es also keine Trennebene gibt, um sie zu separieren, suchen wir nach der Ebene, die den geringsten Fehler verursacht. Wir definieren *Schlupfvariablen*, $\xi^t \geq 0$, um die Abweichung vom definierten Margin zu speichern. Es existieren zwei Arten von Abweichung: Eine Instanz kann auf der falschen Seite der Hyperebene liegen und fehlklassifiziert worden sein. Oder sie liegt zwar auf der richtigen Seite, jedoch innerhalb des Margin, sprich, nicht weit genug von der Hyperebene entfernt. Wir lockern die Bedingungen aus Gleichung 13.1 und verlangen nun

SCHLUPFVARIABLEN

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq 1 - \xi^t \,. \tag{13.9}$$

Für $\xi^t = 0$ gibt es keinerlei Probleme mit $\boldsymbol{x}^t$. Falls $0 < \xi^t < 1$, so ist $\boldsymbol{x}^t$ korrekt klassifiziert, liegt jedoch innerhalb des Margin. Ist $\xi^t \geq 1$, so wurde $\boldsymbol{x}^t$ fehlklassifiziert (siehe Abbildung 13.2). Die Anzahl der Fehlklassifikationen ist $\#\{\xi^t > 1\}$, und die Anzahl der nicht trennbaren Punkten ist $\#\{\xi_t > 0\}$. Wir definieren den *weichen Fehler* (engl. *soft error*) als

WEICHER FEHLER

$$\sum_t \xi^t$$

und addieren dies als Strafterm:

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t \,. \tag{13.10}$$

Dabei ist die Nebenbedingung 13.9 zu erfüllen. C ist der Strafterm wie bei jedem Regularisierungsschema, so dass ein Kompromiss zwischen Komplexität, gemessen durch die L_2-Norm des Gewichtsvektors (ähnlich dem Gewichtsabbau bei mehrlagigen Perzeptronen, siehe die Abschnitte 11.9 und 11.10) und Datenfehlanpassung, gemessen durch die Anzahl nicht trennbarer Punkte, geschlossen wird. Man beachte, dass wir nicht nur die fehlklassifizierten Punkte bestrafen, sondern mit dem Ziel einer besseren Generalisierung auch diejenigen im Margin, obwohl letztere während der Testphase korrekt klassifiziert werden würden.

Unter Berücksichtigung der Nebenbedingungen wird der Lagrange-Funktion von Gleichung 13.4 zu

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t - \sum_t \alpha^t \left[r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) - 1 + \xi^t\right] - \sum_t \mu^t\xi^t \,, \tag{13.11}$$

wobei die μ^t die neuen Lagrange-Parameter sind, welche die Positivität von ξ^t gewährleisten. Wenn wir die Ableitungen nach den Parametern bilden und diese gleich 0 setzen, erhalten wir

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_t \alpha^t r^t \boldsymbol{x}^t = 0 \Rightarrow \boldsymbol{w} = \sum_t \alpha^t r^t \boldsymbol{x}^t \,, \tag{13.12}$$

$$\frac{\partial L_p}{\partial w_0} = \sum_t \alpha^t r^t = 0 \,, \tag{13.13}$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0 \,. \tag{13.14}$$

Wegen $\mu^t \geq 0$ folgt aus der letzten dieser Gleichungen $0 \leq \alpha^t \leq C$. Setzen wir dies in Gleichung 13.11 ein, so erhalten wir die duale Form, die wir bezüglich α^t maximieren:

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\boldsymbol{x}^t)^T \boldsymbol{x}^s \tag{13.15}$$

unter

$$\sum_t \alpha^t r^t = 0 \text{ und } 0 \leq \alpha^t \leq C, \ \forall t \,.$$

Wenn wir dies lösen, dann sehen wir, dass wie im trennbaren Fall für Instanzen, die auf der richtigen Seite der Trennlinie liegen und einen hinreichend großen Abstand haben, $\alpha^t = 0$ ist (siehe Abbildung 13.2). Für die Support-Vektoren gilt $\alpha^t > 0$ und sie definieren gemäß Gleichung 13.12 $\boldsymbol{w}$. Von diesen sind es diejenigen mit $\alpha^t < C$, welche auf dem Margin liegen, und wir können sie verwenden, um w_0 zu berechnen. Für sie gilt $\xi^t = 0$ und sie erfüllen die Bedingung $r^t(\boldsymbol{w}^T \boldsymbol{x}^t + w_0) = 1$. Auch hier ist es wieder besser, einen Mittelwert über diese w_0-Schätzungen zu nehmen. Für die Instanzen, die innerhalb des Margins liegen oder fehlklassifiziert wurden, ist $\alpha^t = C$.

Die nicht trennbaren Instanzen, die wir als Support-Vektoren speichern, sind diejenigen Instanzen, für die es uns Probleme bereiten würde, sie korrekt zu klassifizieren, wenn sie nicht in der Trainingsmenge wären. Wir würden sie entweder falsch klassifizieren oder aber korrekt, jedoch mit nicht ausreichender Konfidenz. Wir können sagen, dass die Anzahl der Support-Vektoren ein oberer Schätzer für die erwartete Anzahl der Fehler ist. Und tatsächlich hat Vapnik (1995) gezeigt, dass für die erwartete Testfehlerrate gilt

$$E_N[P(\text{Fehler})] \leq \frac{E_N[\#\text{ der Support-Vektoren}]}{N} \,.$$

Dabei bezeichnet $E_N[\cdot]$ den Erwartungswert über Trainingsmengen der Größe N. Hieraus folgt die nützliche Eigenschaft, dass die Fehlerrate von

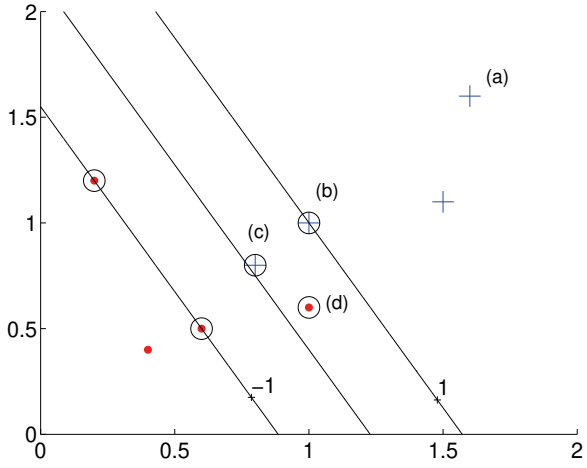


***Abb. 13.2*:** Bei der Klassifikation einer Instanz gibt es vier mögliche Fälle: In Fall (a) liegt die Instanz auf der richtigen Seite und weit weg vom Margin, d. h., es gilt $r^t g(\boldsymbol{x}^t) > 1$ und $\xi^t = 0$. In Fall (b) ist $\xi^t = 0$; sie liegt auf der rechten Seite und auf der Margin-Linie. In Fall (c) ist $\xi^t = 1 - g(\boldsymbol{x}^t)$ und $0 < \xi < 1$; sie liegt auf der rechten Seite, aber innerhalb des Margins und nicht weit genug weg. In Fall (d) ist $\xi^t = 1 + g(\boldsymbol{x}^t) > 1$; sie liegt auf der falschen Seite – dies ist eine Fehlklassifikation. Alle Fälle außer (a) sind Support-Vektoren. Ausgedrückt durch die duale Variable gilt für (a) $\alpha^t = 0$, für (b) $\alpha^t < C$, für (c) und (d) $\alpha^t = C$.

der Anzahl der Support-Vektoren und nicht von der Dimensionalität der Eingabe abhängt.

Gleichung 13.9 impliziert, dass wir es als Fehler definieren, wenn die Instanz auf der falschen Seite liegt oder wenn der Margin kleiner ist als 1. Dies wird als *Scharnierverlust* bezeichnet. Wenn $y^t = \boldsymbol{w}^T \boldsymbol{x}^t + w_0$ die Ausgabe ist und r^t die gewünschte Ausgabe, dann ist der Scharnierverlust definiert als

SCHARNIERVERLUST

$$L_{\text{Scharnier}}(y^t, r^t) = \begin{cases} 0 & \text{falls } y^t r^t \geq 1\,, \\ 1 - y^t r^t & \text{sonst}\,. \end{cases} \tag{13.16}$$

In Abbildung 13.3 wird der Scharnierverlust mit dem 0/1-Verlust, dem quadratischen Fehler und der Kreuzentropie vergleichen. Wir sehen, dass der Scharnierverlust, anders als der 0/1-Verlust, auch Instanzen innerhalb des Margins bestraft, selbst wenn sie auf der korrekten Seite liegen, und der Verlust wächst linear, wenn sich die Instanz auf der falschen Seite weiter entfernt. Dies ist ein Unterschied gegenüber dem quadratischen Verlust, der deshalb nicht so robust ist wie der Scharnierverlust. Wir sehen, dass die Kreuzentropie, die bei der logistischen Diskriminanz

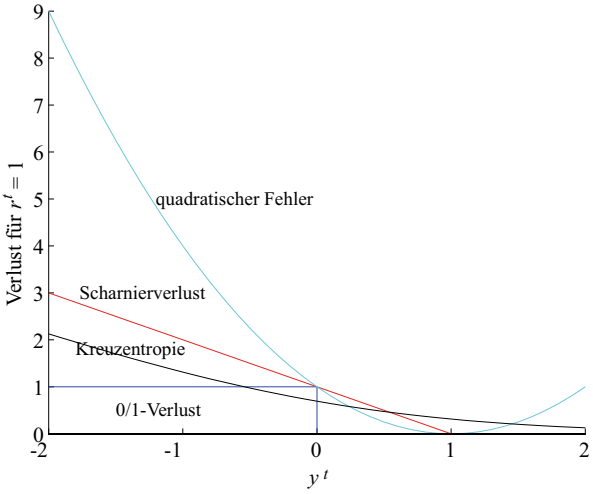


***Abb. 13.3*:** Vergleich verschiedener Verlustfunktionen für $r^t = 1$: Der 0/1-Verlust ist 0, falls $y^t = 1$, andernfalls 1. Der Scharnierverlust ist 0, falls $y^t > 1$, andernfalls $1 - y^t$. Der quadratische Fehler ist $(1 - y^t)^2$. Die Kreuzentropie ist $\log(1/(1 + \exp(-y^t)))$.

(Abschnitt 10.7) oder durch das lineare Perzeptron (Abschnitt 11.3) minimiert wird, eine gute Näherung an den Scharnierverlust ist.

Das C in Gleichung 13.10 ist der durch Kreuzvalidierung feinabgestimmte Regularisierungsparameter. Dieser definiert den Kompromiss zwischen Margin-Maximierung und Fehlerminimierung: Ist er zu groß, haben wir eine hohe Strafe für nicht trennbare Punkte, was zu einer großen Zahl von Support-Vektoren und zu einer Überanpassung führen kann. Wenn er dagegen zu klein ist, kann es sein, dass wir zu einfache Lösungen finden, die eine Unteranpassung darstellen. Typischerweise wählt man aus $[10^{-6}, 10^{-5}, \ldots, 10^{+5}, 10^{+6}]$ auf der logarithmischen Skala, indem man sich die Genauigkeit auf einer Validierungsmenge ansieht.

13.4 v-SVM

Es gibt eine andere, äquivalente Formulierung der Soft-Margin-Trennebene, bei der ein Parameter $v \in [0, 1]$ anstelle von C verwendet wird (Schölkopf et al. 2000). Die Zielfunktion ist

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 - v\rho + \frac{1}{N}\sum_t \xi^t \tag{13.17}$$

unter

$$r^t(\boldsymbol{w}^T\boldsymbol{x}^t + w_0) \geq \rho - \xi^t,\ \xi^t \geq 0,\ \rho \geq 0\,. \tag{13.18}$$

ρ ist ein neuer Parameter, der eine Variable des Optimierungsproblems ist und den Margin skaliert: Der Margin ist nun $2\rho/\|\boldsymbol{w}\|$. Es wurde gezeigt, dass v eine untere Schranke für den Anteil der Support-Vektoren ist und eine obere Schranke für den Anteil der Instanzen, die Margin-Fehler haben ($\sum_t \#\{\xi^t > 0\}$). Die duale Form ist

$$L_d = -\frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s (\boldsymbol{x}^t)^T \boldsymbol{x}^s \tag{13.19}$$

unter

$$\sum_t \alpha^t r^t = 0,\ 0 \leq \alpha^t \leq \frac{1}{N},\ \sum_t \alpha^t \geq v\,.$$

Wenn wir Gleichung 13.19 und Gleichung 13.15 miteinander vergleichen, dann sehen wir, dass der Term $\sum_t \alpha^t$ in der Zielfunktion nicht mehr auftritt, dafür aber als Nebenbedingung. Indem wir an v drehen, können wir den Anteil der Support-Vektoren kontrollieren, was sich als intuitivere Heransgehensweise als ein ein Drehen an C empfiehlt.

13.5 Kernel-Trick

In Abschnitt 10.2 wurde gezeigt, dass wir ein nichtlineares Problem durch eine nichtlineare Transformation in einen neuen Raum abbilden können, anstatt zu versuchen, ein nichtlineares Modell anzupassen. Dazu wählen wir geeignete Basisfunktionen und verwenden dann in diesem neuen Raum ein lineares Modell. Das lineare Modell im neuen Raum entspricht einem nichtlinearen Modell im Originalraum. Dieser Ansatz funktioniert sowohl bei der Klassifikation als auch bei der Regression. Speziell bei der Klassifikation kann er mit jedem beliebigen Schema verwendet werden, wobei er für Support-Vektor-Maschinen zu gewissen Vereinfachungen führt, die wir im Folgenden diskutieren wollen.

Nehmen wir an, wir haben die neuen Dimensionen mithilfe der Basisfunktionen

$$\boldsymbol{z} = \boldsymbol{\phi}(\boldsymbol{x}) \text{ mit } z_j = \phi_j(\boldsymbol{x}),\ j = 1, \ldots, k$$

berechnet, die aus dem d-dimensionalen $\boldsymbol{x}$-Raum in den k-dimensionalen $\boldsymbol{z}$-Raum abbilden, in dem wir die Diskriminanzfunktion als

$$\begin{aligned} g(\boldsymbol{z}) &= \boldsymbol{w}^T \boldsymbol{z}\,, \\ g(\boldsymbol{x}) &= \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) \\ &= \sum_{j=1}^{k} w_j \phi_j(\boldsymbol{x}) \end{aligned} \tag{13.20}$$

schreiben. Wir verwenden hier kein separates w_0 und nehmen an, dass $z_1 = \phi_1(\boldsymbol{x}) \equiv 1$. Allgemein ist k viel größer als d und es kann auch größer sein als N. Genau hierin liegt der Vorteil bei der Verwendung der dualen Form, deren Komplexität von N abhängt, denn würden wir die primale Form verwenden, hätten wir eine von k abhängige Komplexität. Wir verwenden hier außerdem den allgemeineren Ansatz der Soft-Margin-Hyperebene, da wir nicht sicher sein können, ob das Problem in diesem neuen Raum linear trennbar ist.

Das Problem ist das gleiche, nämlich

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t \,, \tag{13.21}$$

mit dem Unterschied, dass die Nebenbedingungen im neuen Raum definiert werden:

$$r^t \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{w}^t) \geq 1 - \xi^t \tag{13.22}$$

Die Lagrange-Funktion ist

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t - \sum_t \alpha^t \left[r^t \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{w}^t) - 1 + \xi^t\right] - \sum_t \mu^t \xi^t . \tag{13.23}$$

Wenn wir die Ableitungen nach den Parametern bilden und sie 0 setzen, erhalten wir

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} = \sum_t \alpha^t r^t \boldsymbol{\phi}(\boldsymbol{x}^t) \,, \tag{13.24}$$

$$\frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0 \,. \tag{13.25}$$

Die duale Form ist nun

$$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s \boldsymbol{\phi}(\boldsymbol{x}^t)^T \boldsymbol{\phi}(\boldsymbol{x}^s) \tag{13.26}$$

unter

$$\sum_t \alpha^t r^t = 0 \text{ und } 0 \leq \alpha^t \leq C, \ \forall t$$

KERNEL-FUNKTION

Die Idee der *Kernel-Maschinen* besteht darin, das Skalarprodukt der Basisfunktionen, $\boldsymbol{\phi}(x^t)^T \boldsymbol{\phi}(x^s)$, durch eine *Kernel-Funktion* $K(\boldsymbol{x}^t, \boldsymbol{x}^s)$ zwischen den Instanzen des originalen Eingaberaums zu ersetzen. Anstatt also zwei Instanzen $\boldsymbol{x}^t$ und $\boldsymbol{x}^s$ in den $\boldsymbol{z}$-Raum abzubilden und dort

ein Skalarprodukt zu bilden, wenden wir die Kernel-Funktion direkt im Originalraum an:

$$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t \sum_s \alpha^t \alpha^s r^t r^s K(\boldsymbol{x}^t, \boldsymbol{x}^s) \,. \tag{13.27}$$

Die Kernel-Funktion tritt auch in der Diskriminanzfunktion auf:

$$\begin{aligned} g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) &= \sum_t \alpha^t r^t \boldsymbol{\phi}(\boldsymbol{x}^t)^T \boldsymbol{\phi}(\boldsymbol{x}) \\ &= \sum_t \alpha^t r^t K(\boldsymbol{x}^t, \boldsymbol{x}) \,. \end{aligned} \tag{13.28}$$

Wenn wir die Kernel-Funktion haben, müssen wir sie also gar nicht in den neuen Raum abbilden. Tatsächlich existiert für jeden gültigen Kernel eine zugehörige Mapping-Funktion, aber es kann viel einfacher sein, $K(\boldsymbol{x}^t, \boldsymbol{x})$ zu verwenden, anstatt $\boldsymbol{\phi}^t(\boldsymbol{x}^t)$, $\boldsymbol{\phi}(\boldsymbol{x})$ zu berechnen und das Skalarprodukt zu bilden. Für viele Algorithmen wurden *Kernel-Versionen* entwickelt, wie wir in den folgenden Abschnitten sehen werden, und das ist der Grund für die Bezeichnung „Kernel-Maschinen“.

KERNEL-VERSION

Die Matrix $\mathbf{K}$ der Kernel-Werte $\mathbf{K}_{ts} = K(\boldsymbol{x}^t, \boldsymbol{x}^s)$ ist die sogenannte *Gram-Matrix*; sie sollte symmetrisch und positiv semidefinit sein. In jüngster Zeit ist es beim gemeinsamen Nutzen von Datenmengen zum Standard geworden, nur die $\mathbf{K}$-Matrizen verfügbar zu machen und auf das Bereitstellen von $\boldsymbol{x}^t$ oder $\boldsymbol{\phi}(\boldsymbol{x}^t)$ zu verzichten. Besonders in der Bioinformatik oder bei Anwendungen der natürlichen Sprachverarbeitung, wo $\boldsymbol{x}$ (oder $\boldsymbol{\phi}(\boldsymbol{x})$) Hunderte oder Tausende Dimensionen hat, ist das Speichern bzw. Herunterladen der $N \times N$-Matrix viel billiger (Vert, Tsuda und Schölkopf 2004). Das bedeutet allerdings, dass wir diese verfügbaren Daten nur zum Trainieren und Testen verwenden können; Vorhersagen außerhalb dieser Datenmenge sind mit dem trainierten Modell dagegen nicht möglich.

GRAM-MATRIX

13.6 Vektorielle Kernel

Die populärsten universell einsetzbaren Kernel-Funktionen sind:

- *Polynome* vom Grad q:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = (\boldsymbol{x}^T \boldsymbol{x}^t + 1)^q \tag{13.29}$$

  Der Grad q kann dabei vom Anwender gewählt werden. Für $q = 2$ und $d = 2$ erhalten wir zum Beispiel

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{y}) &= (\boldsymbol{x}^T \boldsymbol{y} + 1)^2 \\ &= (x_1 y_1 + x_2 y_2 + 1)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 \,, \end{aligned}$$

was dem Skalarprodukt der Basisfunktionen entspricht (Cherkassky und Mulier 1998):

$$\phi(\boldsymbol{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]^T .$$

Ein Beispiel ist in Abbildung 13.4 zu sehen. Für $q = 1$ haben wir den *linearen Kernel*, der der urspünglichen Formulierung entspricht.

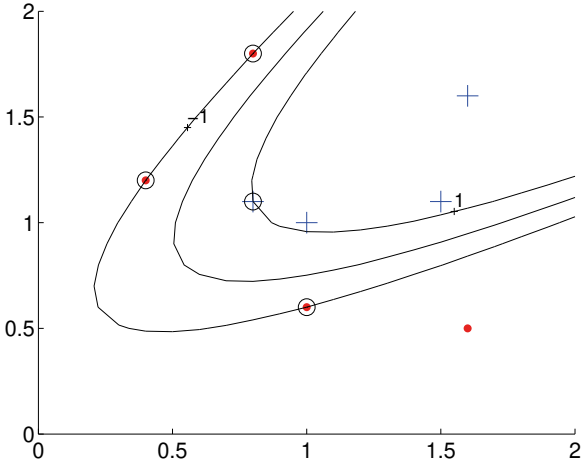


***Abb. 13.4*:** Diskriminanzfunktion und Margins, die durch einen polynomialen Kernel vom Grad 2 gefunden wurden. Die durch Kreise markierten Instanzen sind die Support-Vektoren.

- *Radial-Basisfunktionen:*

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{\|\boldsymbol{x}^t - \boldsymbol{x}\|^2}{2s^2}\right] \tag{13.30}$$

Hierdurch ist ein sphärischer Kernel wie bei den Parzen-Fenstern (Kapitel 8) definiert, wobei $\boldsymbol{x}^t$ der Mittelpunkt ist und s, bereitgestellt vom Anwender, der Radius. Dies ähnelt auch den radialen Basisfunktionen, die wir in Kapitel 12 diskutiert haben.

Ein Beispiel ist in Abbildung 13.5 gezeigt. Hier sehen wir, dass größere Streuungen die Grenze glätten; der beste Wert wird durch Kreuzvalidierung gefunden. Wenn es zwei Parameter gibt, die mittels Kreuzvalidierung zu optimieren sind, hier zum Beispiel C und s^2, dann sollte eine (faktorielle) Gittersuche in zwei Dimensionen durchgeführt werden. In Abschnitt 19.2 werden wir Verfahren für die Suche nach der besten Kombination solcher Faktoren behandeln.

Durch Verallgemeinerung des Euklidischen Abstands kann man einen Mahalanobis-Kernel konstruieren:

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{1}{2}(\boldsymbol{x}^t - \boldsymbol{x})^T\mathbf{S}^{-1}(\boldsymbol{x}^t - \boldsymbol{x})\right] . \tag{13.31}$$

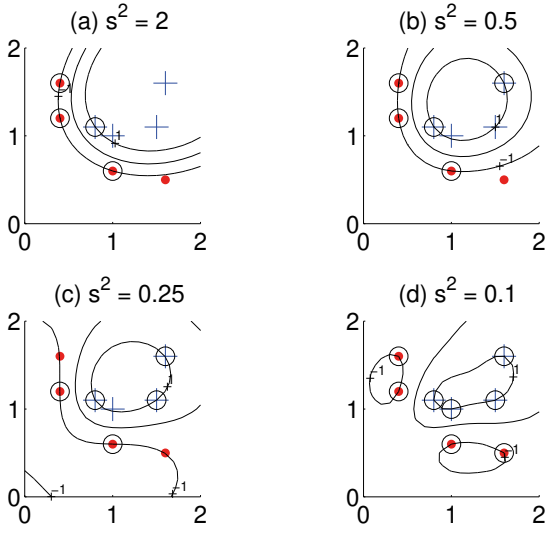


Abb. 13.5: Grenzen und Margins, die durch einen Gauß-Kernel mit unterschiedlichen Werten der Streuung, s^2, gefunden wurden. Für größere Streuwerte erhalten wir glattere Grenzen.

Hierbei ist $\mathbf{S}$ eine Kovarianzmatrix. Der allgemeinste Fall ist

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \exp\left[-\frac{\mathcal{D}(\boldsymbol{x}^t, \boldsymbol{x})}{2s^2}\right] \tag{13.32}$$

mit einer Abstandsfunktion $\mathcal{D}(\boldsymbol{x}^t, \boldsymbol{x})$.

- *Sigmoidale Funktionen:*

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \tanh(2\boldsymbol{x}^T\boldsymbol{x}^t + 1) \tag{13.33}$$

Hier hat $\tan(\cdot)$ die gleiche Form wie die Sigmoidfunktion, mit dem Unterschied, das ihr Wertebereich von -1 bis $+1$ geht. Dies ist ähnlich wie bei den mehrlagigen Perzeptronen, die wir in Kapitel 11 diskutiert haben.

13.7 Definition von Kerneln

Es ist auch möglich, anwendungsspezifische Kernel zu definieren. Kernel werden allgemein als Maße für Ähnlichkeit aufgefasst, d. h., $K(\boldsymbol{x}, \boldsymbol{y})$ nimmt einen umso größeren Wert an, je „ähnlicher“ sich $\boldsymbol{x}$ und $\boldsymbol{y}$ sind. Was unter „ähnlich“ zu verstehen ist, hängt dabei von der Anwendung ab. Das bedeutet, dass jedes Vorwissen, das wir in Bezug auf die Anwendung haben, durch geeignet definierte Kernel für den Lerner bereitgestellt werden

kann. Dieser als Kernel Engineering bezeichnete Gebrauch von Kerneln kann als ein weiteres Beispiel für einen „Hinweis“ (Abschnitt 11.8.4) angesehen werden.

Es gibt Zeichenketten-Kernel, Baum-Kernel, Graphen-Kernel usw. (Vert, Tsuda und Schölkopf 2004), je nachdem, wie wir die Daten repräsentieren und wie wir in dieser Repräsentationen Ähnlichkeit messen.

Bag of Words

Wenn beispielsweise zwei Dokumente gegeben sind, kann die Anzahl der Wörter, die in beiden Dokumenten vorkommen, ein Kernel sein. Bezeichnen wir die beiden Dokumente mit D_1 und D_2. Eine mögliche Repräsentation ist das *Bag-of-Words-Modell,* bei dem wir M Wörter vordefinieren, die für die Anwendung relevant sind. Wir definieren nun $\boldsymbol{\phi}(D_1)$ als den M-dimensionalen binären Vektor, dessen Komponente i gleich 1 ist, wenn das Wort i in D_1 vorkommt, und sonst 0. Dann gibt $\boldsymbol{\phi}(D_1)^T\boldsymbol{\phi}(D_2)$ die Anzahl der in beiden Dokumenten vorkommenden Wörter an. Wenn wir $K(D_1, D_2)$ direkt als die Anzahl der gemeinsamen Wörter definieren und implementieren, ist es offensichtlich nicht nötig, dass wir M Wörter vorselektieren. Wir können einfach ein beliebiges Wort des Vokabulars verwenden (selbstverständlich, nachdem Wörter wie „und“, „von“ usw. entfernt wurden) und müssen die Bag-of-Words-Repräsentation nicht explizit generieren; es wäre so, als hätten wir erlaubt, dass M so groß ist, wie wir wollen.

Editierdistanz

Alignment

Manchmal – zum Beispiel in Anwendungen der Bioinformatik – können wir einen *Ähnlichkeitsscore* für zwei Objekte berechnen, der nicht notwendigerweise positiv semidefinit ist. Wenn wir zwei Zeichenketten (für Gensequenzen) gegeben haben, dann misst ein Kernel die *Editierdistanz,* d. h., die Anzahl der nötigen Operationen (Einfügen, Entfernen, Substitution), um eine Zeichenkette in eine andere zu überführen. Dies wird auch als *Alignment* bezeichnet. In einem solchen Fall kann man einen Trick anwenden, der darin besteht, eine Menge von M Templates zu definieren und ein Objekt als den M-dimensionalen Vektor der Treffer mit allen diesen Templates darzustellen. Das heißt, wenn die $\boldsymbol{m}_i$ mit $i = 1, \ldots, M$ die Templates sind und $s(\boldsymbol{x}^t, \boldsymbol{m}_i)$ der Wert für den Treffer zwischen $\boldsymbol{x}^t$ und $\boldsymbol{m}_i$, dann definieren wir

$$\boldsymbol{\phi}(\boldsymbol{x}^t) = \left[s(\boldsymbol{x}^t, \boldsymbol{m}_1), s(\boldsymbol{x}^t, \boldsymbol{m}_2), \ldots, s(\boldsymbol{x}^t, \boldsymbol{m}_M)\right]^T .$$

empirische Kernel-Map

Damit definieren wir die *empirische Kernel-Map* als

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x}^t)^T\boldsymbol{\phi}(\boldsymbol{x}^s) ,$$

was ein gültiger Kernel ist.

Manchmal haben wir eine binäre Trefferfunktion. Beispielsweise können zwei Proteine miteinander interagieren oder nicht, und wir würden gern in der Lage sein, dies auf Treffer zwischen zwei beliebigen Instanzen zu verallgemeinern. In einem solchen Fall besteht der Trick darin, einen

Graphen zu definieren, dessen Knoten die Instanzen sind. Zwei Knoten sind miteinander verbunden, wenn sie miteinander interagieren, d. h. wenn die binäre Trefferfunktion 1 ist. Dann sagen wir, dass zwei nicht direkt miteinander verbundene Knoten sich „ähnlich“ sind, wenn der Weg zwischen ihnen kurz ist oder wenn sie durch viele Wege miteinander verbunden sind. Damit überführen wir paarweise lokale Wechselwirkungen in ein globales Ähnlichkeitsmaß, ungefähr so wie bei der Definition des geodätischen Abstands bei Isomap (Abschnitt 6.10). Ein solcher Kernel wird als *Diffusionskernel* bezeichnet.

DIFFUSIONSKERNEL

Wenn $p(\boldsymbol{x})$ eine Wahrscheinlichkeitsdichte ist, dann ist

$$K(\boldsymbol{x}^t, \boldsymbol{x}) = p(\boldsymbol{x}^t)p(\boldsymbol{x})$$

ein gültiger Kernel. Dieser wird verwendet, wenn $p(\boldsymbol{x})$ ein generatives Modell für $\boldsymbol{x}$ ist, dass misst, mit welcher Wahrscheinlichkeit wir $\boldsymbol{x}$ sehen. Wenn $\boldsymbol{x}$ zum Beispiel eine Sequenz ist, kann $p(\boldsymbol{x})$ ein Hidden-Markov-Modell sein (Kapitel 15). Dieser Kernel $K(\boldsymbol{x}^t, \boldsymbol{x})$ nimmt einen hohen Wert an, wenn es wahrscheinlich ist, dass $\boldsymbol{x}^t$ und $\boldsymbol{x}$ beide durch das gleiche Modell generiert wurden. Es ist auch möglich, das generative Modell in der Form $p(\boldsymbol{x}|\theta)$ zu parametrisieren und θ aus den Daten zu lernen. Dies wird als *Fisher-Kernel* bezeichnet (Jaakkola und Haussler 1998).

FISHER-KERNEL

13.8 Multiple-Kernel-Lernen

Es ist möglich, neue Kernel durch Kombination einfacherer Kernel zu konstruieren. Wenn $K_1(\boldsymbol{x}, \boldsymbol{y})$ und $K_2(\boldsymbol{x}, \boldsymbol{y})$ gültige Kernel sind und c eine Konstante ist, dann ist

$$K(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} cK_1(\boldsymbol{x}, \boldsymbol{y}) \\ K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y}) \\ K_1(\boldsymbol{x}, \boldsymbol{y}) \cdot K_2(\boldsymbol{x}, \boldsymbol{y}) \end{cases} \tag{13.34}$$

ebenfalls ein gültiger Kernel.

Unterschiedliche Kernel können auch unterschiedliche Teilmengen von $\boldsymbol{x}$ verwenden. Wir können daher das Kombinieren von Kerneln als eine weitere Möglichkeit betrachten, Informationen aus verschiedenen Quellen zu vereinigen, wobei jeder Kernel die Ähnlichkeit entsprechend seines

Bereichs misst. Dann haben wir eine Eingabe aus zwei Repräsentationen A und B

$$\begin{aligned} K_A(\boldsymbol{x}_A, \boldsymbol{y}_A) + K_B(\boldsymbol{x}_B, \boldsymbol{y}_B) &= \boldsymbol{\Phi}_A(\boldsymbol{x}_A)^T \boldsymbol{\Phi}_A(\boldsymbol{y}_A) + \boldsymbol{\Phi}_B(\boldsymbol{x}_B)^T \boldsymbol{\Phi}_B(\boldsymbol{y}_B) \\ &= \boldsymbol{\Phi}(\boldsymbol{x})^T \boldsymbol{\Phi}(\boldsymbol{y}) \\ &= K(\boldsymbol{x}, \boldsymbol{y}) \,, \end{aligned} \tag{13.35}$$

wobei $\boldsymbol{x} = [\boldsymbol{x}_A, \boldsymbol{x}_B]$ die Konkatenation der beiden Repräsentationen ist. Das bedeutet, dass das Summieren zweier Kernel dem Bilden des Skalarprodukts der konkatenierten Merkmalsvektoren ist. Man kann dies auf mehrere Kernel verallgemeinern:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} K_i(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{13.36}$$

Dies ähnelt der Bildung des Mittelwerts für einen Klassifikator (Abschnitt 17.4), wobei wir diesmal über Kernel mitteln und somit die Notwendigkeit entfällt, einen bestimmten Kernel auszuwählen. Es ist auch möglich, eine gewichtete Summe zu nehmen und die Gewichte aus den Daten zu lernen (Lanckriet et al. 2004; Sonnenburg et al. 2006):

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} \eta_i K_i(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{13.37}$$

Dabei ist $\eta_i \geq 0$ zu erfüllen, mit oder ohne die Nebenbedingung $\sum_i \eta_i = 1$, was als konvexe bzw. konische Kombination bekannt ist. Dieses Verfahren, bei dem ein einzelner Kernel durch eine gewichtete Summe ersetzt wird, wird als Multiple-Kernel-Lernen (Lernen mit mehreren Kernen) bezeichnet (Gönen und Alpaydın 2011). Die Zielfunktion für einen einzelnen Kernel, Gleichung 13.27, wird dann zu

Multiple-Kernel-Lernen

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \sum_i \eta_i K_i(\boldsymbol{x}^t, \boldsymbol{x}^s) \,, \tag{13.38}$$

was wir nach den Parametern α^t der Support-Vektor-Maschine und den Kernel-Gewichten η_i auflösen. Die Kombination mehrerer Kernel erscheint dann auch in der Diskriminanzfunktion

$$g(\boldsymbol{x}) = \sum_t \alpha^t r^t \sum_i \eta_i K_i(\boldsymbol{x}^t, \boldsymbol{x}) \,. \tag{13.39}$$

Nach dem Training wird η_i Werte annehmen, die davon abhängen, wie nützlich der zugehörige Kernel $K_i(\boldsymbol{x}^t, \boldsymbol{x})$ für die Unterscheidung ist. Es ist auch möglich, Kernel zu lokalisieren, indem die Kernel-Gewichte als parametrisierte Funktionen der Eingabe $\boldsymbol{x}$ definiert werden, ganz ähnlich wie bei der Gatterfunktion in gemischten Expertensystemen (Abschnitt 17.8):

$$g(\boldsymbol{x}) = \sum_t \alpha^t r^t \sum_i \eta_i(\boldsymbol{x}|\theta_i) K_i(\boldsymbol{x}^t, \boldsymbol{x}) \,. \tag{13.40}$$

Die Gatterparameter θ_i werden zusammen mit den Parametern der Support-Vektor-Maschine gelernt (Gönen und Alpaydın 2008).

Wenn wir Informationen aus verschiedenen Quellen haben, die jeweils in unterschiedlichen Repräsentationen oder Formen vorliegen – bei der Spracherkennung zum Beispiel in Form von akustischen Informationen und Bildern der Lippen – dann ist der gewöhnliche Ansatz der, diese zunächst mit verschiedenen Klassifikatoren separat zu verarbeiten und die Entscheidungen dann miteinander zu verschmelzen. In Kapitel 17 werden wir solche Verfahren genauer untersuchen. Die Kombination mehrerer Kernel bietet uns eine weitere Möglichkeit, um Eingaben aus unterschiedlichen Quellen zu integrieren. Dabei gibt es einen einzigen Klassifikator, der verschiedene Kernel für die Eingaben aus den verschiedenen Quellen verwendet, und diese arbeiten mit unterschiedlichen Ähnlichkeitsbegriffen (Noble 2004). Die lokalisierte Version kann dann als eine Erweiterung dieses Ansatzes betrachtet werden, bei der wir die Quellen und folglich die Ähnlichkeitsmaße in Abhängigkeit von der Eingabe wählen können.

13.9 Mehrklassen-Kernel-Maschinen

Für $K > 2$ Klassen ist der naheliegende Weg *einer-vs.-alle,* d. h., es werden K Zweiklassen-Probleme definiert, die jeweils eine Klasse von der Kombination aller anderen separieren und K Support-Vektor-Maschinen $g_i(\boldsymbol{x}), i = 1, \ldots, K$, lernen. Das heißt, beim Trainieren der $g_i(\boldsymbol{x})$ bekommen Beispielinstanzen von C_i das Label $+1$ und Beispielinstanzen von C_k mit $k \neq i$ bekommen das Label -1. Während des Testens berechnen wir alle $g_i(\boldsymbol{x})$ und wählen das Maximum.

Platt (1999) hat vorgeschlagen, eine Sigmoidfunktion an die Ausgabe einer einzelnen (Zweiklassen-)SVM anzupassen, um eine a-posteriori-Wahrscheinlichkeit zu bekommen. Ebenso kann man eine Schicht von Softmax-Ausgaben darauf trainieren, die Kreuzentropie zu minimieren, um $K > 2$ a-posteriori-Wahrscheinlichkeiten zu generieren (Mayoraz und Alpaydın 1999):

$$y_i(\boldsymbol{x}) = \sum_{j=1}^{K} v_{ij} f_j(\boldsymbol{x}) + v_{i0} \,. \tag{13.41}$$

Dabei sind die $f_j(\boldsymbol{x})$ die SVM-Ausgaben und y_i die a-posteriori-Ausgaben. Die Gewichte v_{ij} werden so trainiert, dass sie die Kreuzentropie minimieren. Um die Überanpassung zu mildern, ist allerdings zu beachten, dass – wie bei der Schachtelung (Abschnitt 17.9) – die Daten, auf denen wir die v_{ij} trainieren, andere sein müssen als die Daten, die für das Trainieren der Basis-SVMs $f_j(\boldsymbol{x})$ verwendet wurden.

Anstelle der üblichen Vorgehensweise, K Zweiklassen-SVM-Klassifikatoren zu konstruieren, um, wie mit jedem beliebigen anderen Klassifikator,

jeweils eine Klasse von den übrigen zu separieren, kann man auch $K(K-1)/2$ *paarweise* Klassifikatoren konstruieren (siehe auch Abschnitt 10.4), wobei jedes $g_{ij}(\boldsymbol{x})$ Beispiele von C_i mit dem Label $+1$ und Beispiele von C_i mit dem Label -1 nimmt, aus den übrigen Klassen jedoch keine Beispiele. Das paarweise Separieren ist eine Aufgabe, von der man normalerweise annehmen kann, dass sie einfacher ist, und die den zusätzlichen Vorteil hat, dass die Optimierung wegen des geringeren Umfangs der verwendeten Daten schneller sein wird, auch wenn festzuhalten ist, dass wir $\mathcal{O}(K^2)$ Diskriminanzfunktionen trainieren müssen anstatt $\mathcal{O}(K)$.

FEHLERKORREKTUR-CODES

Sowohl die Separation „einer-vs.-alle“ als auch die paarweise Separation sind Spezialfälle von allgemeinen *Fehlerkorrekturcodes* (engl. *error-correcting output codes,* Abk. ECOC), die ein Mehrklassen-Problem in ein Zweiklassen-Problem zerlegen (Dietterich und Bakiri 1995) (siehe auch Abschnitt 17.6). SVMs sind als Zweiklassen-Klassifikatoren ideal hierfür geeignet (Allwein, Schapire und Singer 2000), und um eine verbesserte ECOC-Matrix zu bekommen, ist es auch möglich, einen inkrementellen Ansatz zu verfolgen, bei dem neue Zweiklassen-SVMs hinzugefügt werden, um Paare von Klassen besser zu separieren, die unscharf getrennt sind (Mayoraz und Alpaydın 1999).

Eine weitere Möglichkeit besteht darin, ein einzelnes Mehrklassen-Optimierungsproblem unter Einbeziehung aller Klassen zu schreiben (Weston und Watkins 1998):

$$\min \frac{1}{2}\sum_{i=1}^{K}\|\boldsymbol{w}_i\|^2 + C\sum_i\sum_t \xi_i^t \tag{13.42}$$

unter

$$\boldsymbol{w}_{z^t}\boldsymbol{x}^t + w_{z^t 0} \geq \boldsymbol{w}_i\boldsymbol{x}^t + w_{i0} + 2 - \xi_i^t,\ \forall i \neq z^t \text{ und } \xi_i^t \geq 0\,.$$

z^t enthält hierbei den Klassenindex von $\boldsymbol{x}^t$. Die Regularisierungsterme minimieren die Normen aller Hyperebenen gleichzeitig, und die Nebenbedingungen stellen sicher, dass der Margin zwischen der tatsächlichen Klasse und allen anderen mindestens 2 ist. Die Ausgabe für die richtige Klasse sollte mindestens $+1$ sein, die Ausgabe aller anderen Klassen sollte mindestens -1 sein, und die Schlupfvariablen sind so definiert , dass sie die Unterschiede ausgleichen.

Obwohl das alles recht vernünftig aussieht, wird der Ansatz „einer-vs.-alle“ allgemein vorgezogen, weil er K separate Probleme mit N Variablen löst, während es bei der Mehrklassen-Formulierung $K \cdot N$ Variablen gibt.

13.10 Kernel-Maschinen und Regression

Nun wollen wir uns ansehen, wie Support-Vektor-Maschinen für die Verwendung bei der Regression verallgemeinert werden können. Wir

stellen fest, dass dieselbe Vorgehensweise – Definition akzeptabler Margins, Schlupfvariablen und einer Regularisierungsfunktion, die Glattheit und Fehler kombiniert – auch hier anwendbar ist. Wir beginnen mit einem linearen Modell und werden später sehen, wie sich auch in diesem Fall Kernel-Funktionen anwenden lassen:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 \; .$$

Bei der reinen Regression verwenden wir das Quadrat der Differenz als Fehler:

$$e_2(r^t, f(\boldsymbol{x}^t)) = [r^t - f(\boldsymbol{x}^t)]^2 \; .$$

Bei der Support-Vektor-Regression dagegen verwenden wir die ϵ-sensitive Verlustfunktion:

$$e_\epsilon(r^t, f(\boldsymbol{x}^t)) = \begin{cases} 0 & \text{falls } |r^t - f(\boldsymbol{x}^t)| < \epsilon \\ |r^t - f(\boldsymbol{x}^t)| - \epsilon & \text{sonst.} \end{cases} \tag{13.43}$$

Das bedeutet, dass wir Fehler bis zur Schwelle ϵ tolerieren und dass über dieser Schwelle liegende Fehler einen linearen Effekt haben und keinen quadratischen. Aus diesem Grund ist diese Fehlerfunktion toleranter gegenüber Rauschen und insofern *robuster* (siehe Abbildung 13.6). Wie beim Scharnierverlust gibt es einen fehlerfreien Bereich, so dass wir ein dünn besetztes Problem haben.

ROBUSTE REGRESSION

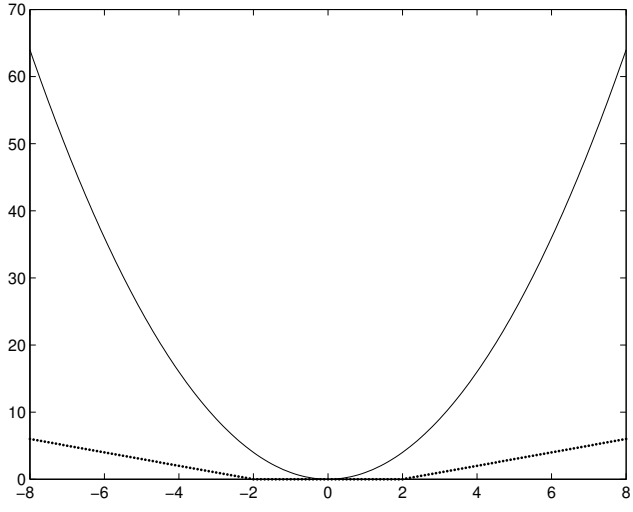


***Abb. 13.6*:** Quadratische und ϵ-sensitive Fehlerfunktionen. Wir sehen, dass kleine Fehler keinen Einfluss auf die ϵ-sensitive Fehlerfunktion haben und diese auch gegenüber großen Fehlern, und somit gegenüber Ausreißern, robuster ist.

Wie bei der Soft-Margin-Hyperebene führen wir Schlupfvariablen ein, um Abweichungen zu berücksichtigen, die aus der ϵ-Zone herausreichen, und erhalten (Vapnik 1995):

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t(\xi_+^t + \xi_-^t) \tag{13.44}$$

unter

$$\begin{aligned} r^t - (\boldsymbol{w}^T\boldsymbol{x} + w_0) &\leq \epsilon + \xi_+^t\,, \\ (\boldsymbol{w}^T\boldsymbol{x} + w_0) - r^t &\leq \epsilon + \xi_-^t\,, \\ \xi_+^t, \xi_-^t &\geq 0\,. \end{aligned}$$

Wir verwenden hier zwei Typen von Schlupfvariablen – einen für positive und einen für negative Abweichungen – um diese positiv zu halten. Tatsächlich können wir uns das Ganze als zwei Scharniere vorstellen, die Rücken an Rücken zusammengefügt sind; das eine für positive und das andere für negative Abweichungen. Diese Formulierung entspricht der durch Gleichung 13.43 gegebenen ϵ-sensitiven Verlustfunktion. Die Lagrange-Funktion ist

$$\begin{aligned} L_p =& \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t(\xi_+^t + \xi_-^t) \\ &- \sum_t \alpha_+^t\left[\epsilon + \xi_+^t - r^t + (\boldsymbol{w}^T\boldsymbol{x} + w_0)\right] \\ &- \sum_t \alpha_-^t\left[\epsilon + \xi_-^t + r^t - (\boldsymbol{w}^T\boldsymbol{x} + w_0)\right] \\ &- \sum_t \left(\mu_+^t\xi_+^t + \mu_-^t\xi_-^t\right)\,. \end{aligned} \tag{13.45}$$

Wir bilden die partiellen Ableitungen und erhalten

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_t(\alpha_+^t - \alpha_-^t)\boldsymbol{x}^t = 0 \Rightarrow \boldsymbol{w} = \sum_t(\alpha_+^t - \alpha_-^t)\boldsymbol{x}^t \tag{13.46}$$

$$\frac{\partial L_p}{\partial w_0} = \sum_t(\alpha_+^t - \alpha_-^t) = 0 \tag{13.47}$$

$$\frac{\partial L_p}{\partial \xi_+^t} = C - \alpha_+^t - \mu_+^t = 0 \tag{13.48}$$

$$\frac{\partial L_p}{\partial \xi_-^t} = C - \alpha_-^t - \mu_-^t = 0\,. \tag{13.49}$$

Die duale Form ist

$$L_d = -\frac{1}{2}\sum_t \sum_s (\alpha_+^t - \alpha_-^t)(\alpha_+^s - \alpha_-^s)(\boldsymbol{x}^t)^T \boldsymbol{x}^s - \epsilon \sum_t (\alpha_+^t + \alpha_-^t) + \sum_t r^t(\alpha_+^t - \alpha_-^t) \tag{13.50}$$

unter

$$0 \le \alpha_+^t \le C,\ 0 \le \alpha_-^t \le C,\ \sum_t (\alpha_+^t - \alpha_-^t) = 0\,.$$

Nachdem wir dies gelöst haben, sehen wir, dass für alle Instanzen innerhalb des Schlauchs $\alpha_+^t = \alpha_-^t = 0$ gilt; das sind die Instanzen, die mit ausreichender Genauigkeit angepasst sind (siehe Abbildung 13.7). Die Support-Vektoren erfüllen entweder $\alpha_+^t > 0$ oder $\alpha_-^t > 0$ und gehören zu einem von zwei Typen. Diese Instanzen können auf dem Rand des Schlauchs liegen (entweder α_+^t oder α_-^t liegt zwischen 0 und C), und wir verwenden sie, um w_0 zu berechnen. Wenn wir zum Beispiel $\alpha_+^t > 0$ annehmen, dann haben wir $r^t = \boldsymbol{w}^T\boldsymbol{x}^t + w_0 + \epsilon$. Instanzen, die außerhalb des ϵ-Schlauchs liegen, gehören zum zweiten Typ; für diese haben wir keine gute Anpassung ($\alpha_+^t = C$), wie man in Abbildung 13.7 sieht.

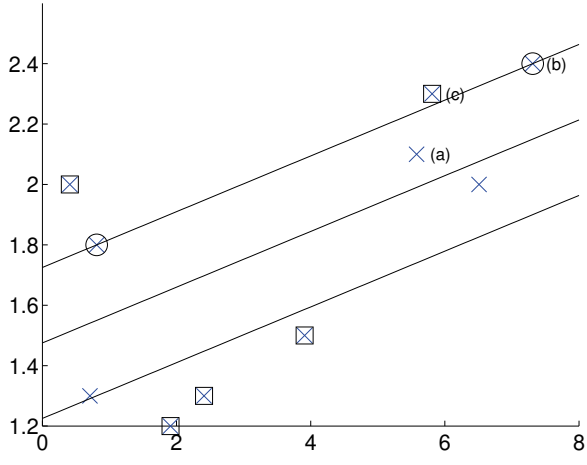


Abb. 13.7: Dargestellt sind die an die Datenpunkte (symbolisiert durch Kreuze) angepasste Regressionslinie und der ϵ-Schlauch ($C = 10, \epsilon = 0{,}25$). Es gibt drei Fälle: In (a) liegt die Instanz innerhalb des Schlauchs; in (b) liegt die Instanz auf dem Rand (durch Kreise gekennzeichnete Instanzen); in (c) liegt sie außerhalb des Schlauchs mit einem positiven Schlupf, d. h. $\xi_+^t > 0$ (durch Quadrate gekennzeichnete Instanzen). (b) und (c) sind Support-Vektoren. Ausgedrückt durch die duale Variable, gilt im Fall (a) $\alpha_+^t = 0, \alpha_-^t = 0$, im Fall (b) $\alpha_+^t < C$ und im Fall (c) $\alpha_+^t = C$.

Mithilfe von Gleichung 13.46 können wir die angepasste Linie als eine gewichtete Summe der Support-Vektoren schreiben:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \sum_t (\alpha_+^t - \alpha_-^t)(\boldsymbol{x}^t)^T\boldsymbol{x} + w_0 \,. \tag{13.51}$$

Wieder kann das Skalarprodukt $(\boldsymbol{x}^t)^T\boldsymbol{x}^s$, welches in Gleichung 13.50 auftritt, durch einen Kernel $K(\boldsymbol{x}^t, \boldsymbol{x}^s)$ ersetzt werden, und wenn wir entsprechend $(\boldsymbol{x}^t)^T\boldsymbol{x}$ durch $K(\boldsymbol{x}^t, \boldsymbol{x})$ ersetzen, können wir eine nichtlineare Anpassung haben. Das Verwenden eines polynomialen Kernels wäre ähnlich wie das Anpassen eines Polynoms (Abbildung 13.8) und das Verwenden eines Gaußschen Kernels (Abbildung 13.9) würde nichtparametrischen Glättungsmodellen (Abschnitt 8.8) ähneln, mit dem Unterschied, dass wir nicht die gesamte Trainingsmenge benötigen würden, sondern nur eine Teilmenge, weil das Problem dünn besetzt ist.

Es gibt auch eine äquivalente v-SVN-Formulierung für die Regression (Schölkopf et al. 2000), bei der v anstatt ϵ festgehalten wird, um den Anteil der Support-Vektoren zu beschränken. Trotzdem wird C weiterhin benötigt.

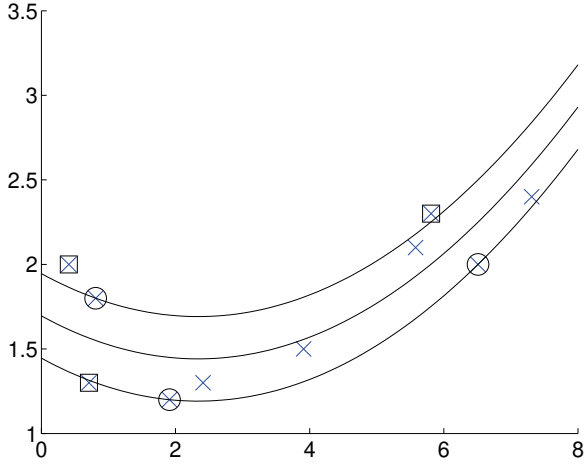


***Abb. 13.8*:** Dargestellt sind die Regressionskurve und der ϵ-Schlauch bei Verwendung eines quadratischen Kernels ($C = 10, \epsilon = 0{,}25$). Die durch Kreise gekennzeichneten Instanzen sind die Support-Vektoren auf den Margins, die mit Quadraten gekennzeichneten Instanzen sind Support-Vektoren, die Ausreißer sind.

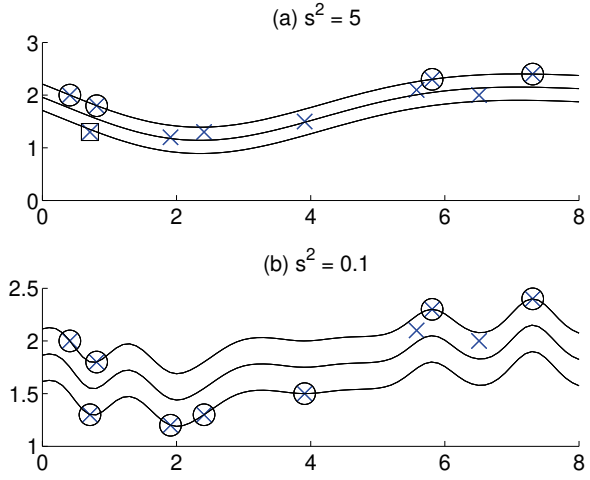


Abb. 13.9: Dargestellt sind die Regressionskurve und der ϵ-Schlauch bei Verwendung eines Gaußschen Kernels mit zwei verschiedenen Werten der Streuung ($C = 10, \epsilon = 0{,}25$). Die durch Kreise gekennzeichneten Instanzen sind die Support-Vektoren auf den Margins, die mit Quadraten gekennzeichneten Instanzen sind Support-Vektoren, die Ausreißer sind.

13.11 Kernel-Maschinen und Ranking

Beim Ranking geht es darum, Instanzen in einer bestimmten Weise zu ordnen (Liu 2011). Beispielsweise können wir paarweise Nebenbedingungen wie $r^u \prec r^v$ haben, was bedeutet, dass die Instanz $\boldsymbol{x}^u$ einen höheren Score generieren soll als $\boldsymbol{x}^v$. In Abschnitt 10.9 haben wir diskutiert, wie wir ein lineares Modell mithilfe des Gradientenabstiegs für diesen Zweck trainieren können. Nun befassen wir uns mit der Frage, wie sich das gleiche mit Support-Vektor-Maschinen erreichen lässt.

Wir betrachten jede paarweise Nebenbedingung als eine Dateninstanz $t : r^u \prec r^v$ und minimieren

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t \tag{13.52}$$

unter

$$\begin{aligned} &\boldsymbol{w}^T\boldsymbol{x}^u \geq \boldsymbol{w}^T\boldsymbol{x}^v + 1 - \xi^t, \text{ für jedes } t : r^u \prec r^v \\ &\xi^t \geq 0\,. \end{aligned} \tag{13.53}$$

Gleichung 13.53 fordert, dass der Score für $\boldsymbol{x}^u$ mindestens eine Einheit größer sein muss als der Score für $\boldsymbol{x}^v$, und definiert somit einen Margin. Wenn die Nebenbedingung nicht erfüllt wird, ist die Schlupfvariable verschieden von null und Gleichung 13.52 minimiert die Schlupfsumme

sowie den Komplexitätsterm, der wieder dafür sorgt, den Margin so breit wie möglich zu machen (Herbrich, Obermayer und Graepel 2000; Joachims 2002). Man beachte, dass der zweite Term der Schlupfsumme dem Fehler gleicht, der in Gleichung 10.46 verwendet wird (bis auf den Margin von einer Einheit), und dass der Komplexitätsterm, den wir zuvor diskutiert haben, als abfallender Gewichtsterm für das lineare Modell interpretiert werden kann (siehe Abschnitt 11.10).

Weiter sei angemerkt, dass es eine Nebenbedingung für jedes Paar gibt, für das eine Ordnung definiert ist; folglich ist die Anzahl solcher Nebenbedingungen $\mathcal{O}(N^2)$. Die Nebenbedingung von Gleichung 13.53 kann auch in der Form

$$\boldsymbol{w}^T(\boldsymbol{x}^u - \boldsymbol{x}^v) \geq 1 - \xi^t$$

geschrieben werden. Das heißt, dass wir dieses Problem als eine Zweiklassen-Klassifikation von paarweisen Differenzen $\boldsymbol{x}^u - \boldsymbol{x}^v$ ansehen können. Indem wir also diese Differenzen berechnen und ihnen $r^t \in \{-1, +1\}$ zuordnen, je nachdem, ob $r^v \prec r^u$ oder $r^u \prec r^v$ gilt, können wir jede Zweiklassen-Kernelmaschine verwenden, um das Ranking-Problem zu implementieren. Dies ist jedoch nicht der effizienteste Weg der Implementierung (siehe Chapelle und Keerthi 2010).

Die duale Form ist

$$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t\sum_s \alpha^t\alpha^s(\boldsymbol{x}^u - \boldsymbol{x}^v)^T(\boldsymbol{x}^k - \boldsymbol{x}^l) \tag{13.54}$$

unter $0 \leq \alpha^t \leq C$. Hierbei sind t und s zwei paarweise Nebenbedingungen der Form $t : r^u \prec r^v$ und $s : r^v \prec r^u$. Wenn wir dies für die Nebenbedingungen, die erfüllt sind, auflösen, haben wir $\xi^t = 0$ und $\alpha^t = 0$. Für diejenigen, die erfüllt sind, aber auf dem Margin liegen, haben wir $0 < \xi^t < 1$ und $\alpha^t < C$, und für die nicht erfüllten (oder falsch eingeordneten) haben wir $\xi^t > 1$ und $\alpha^t = C$.

Für neue Testinstanzen $\boldsymbol{x}$ wird der Score als

$$g(\boldsymbol{x}) = \sum_t \alpha^t(\boldsymbol{x}^u - \boldsymbol{x}^v)^T\boldsymbol{x} \tag{13.55}$$

berechnet. Es ist nicht schwierig, die Kernel-Versionen der primalen und der dualen Form sowie der Score-Funktion aufzuschreiben (Übung 7).

13.12 Einklassen-Kernel-Maschinen

Support-Vektor-Maschinen, die ursprünglich für die Klassifikation vorgeschlagen wurden, konnten auf die Regression ausgedehnt werden, indem anstelle der Diskriminanzfunktion Schlupfvariablen für die Abweichung

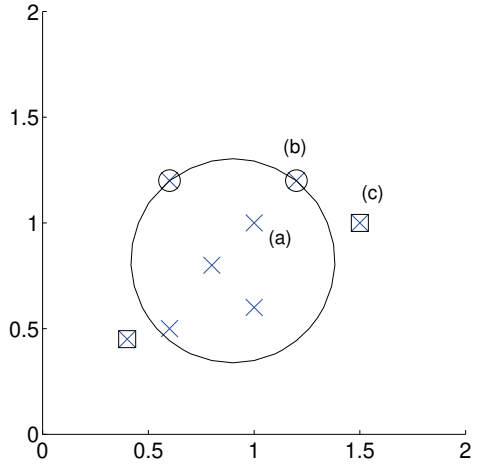


***Abb. 13.10*:** Eine Einklassen-SVM platziert die glatteste Grenze (hier wird ein linearer Kernel, der Kreis mit dem kleinsten Radius, benutzt), die so viele Instanzen wie möglich einschließt. Es gibt drei mögliche Fälle: In (a) ist die Instanz eine typische Instanz. In (b) liegt die Instanz auf der Grenze mit $\xi^t = 0$; solche Instanzen definieren R. In (c) ist die Instanz ein Ausreißer mit $\xi^t > 0$. (b) und (c) sind Support-Vektoren. Ausgedrückt durch die duale Variable haben wir in (a) $\alpha^t = 0$, in (b) $0 < \alpha^t < C$ und in (c) $\alpha^t = C$.

um die Regressionskurve definiert wurden. Nun wollen wir uns ansehen, wie SVMs für einen bestimmten Typ des nicht überwachten Lernens, und zwar das Schätzen von Bereichen hoher Dichte, verwendet werden kann. Unser Ziel ist hier keine vollständige Dichteschätzung, vielmehr wollen wir eine Grenze finden (weshalb das Problem wie eine Klassifikation erscheint), welche Volumen hoher Dichte von Volumen geringer Dichte separiert (Tax und Duin 1999). Eine solche Grenze kann dann für die Erkennung von *Neuheiten* und *Ausreißern* verwendet werden. Dieses Problem wird auch *Einklassen-Klassifikation* genannt.

Ausreisser-erkennung
Einklassen-Klassifikation

Wir betrachten eine Kugel mit Mittelpunkt $\boldsymbol{a}$ und Radius R, die wir so umhüllen wollen, dass so viel wie möglich von der Dichte, empirisch gemessen als der umschlossene Anteil der Trainingsmenge, in der Hülle enthalten ist; gleichzeitig soll der Radius der Hülle so klein wie möglich werden, was im Wettstreit mit der ersten Forderung steht (siehe Abbildung 13.10). Wir definieren Schlupfvariablen für Instanzen, die außerhalb liegen (wir haben nur einen Typ von Schlupfvariablen, weil wir Beispiele von einer Klasse haben und es keine Bestrafung für die innerhalb liegenden gibt), und wir haben ein Glattheitsmaß, das proportional zum Radius ist:

$$\min R^2 + C \sum_t \xi^t \tag{13.56}$$

unter

$$\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 \leq R^2 + \xi^t \text{ und } \xi^t \geq 0, \; \forall t \; .$$

Durch Hinzufügen der Nebenbedingungen erhalten wir die Lagrange-Funktion, die wir unter Berücksichtigung von $\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 = (\boldsymbol{x}^t - \boldsymbol{a})^T(\boldsymbol{x}^t - \boldsymbol{a})$ aufschreiben:

$$\begin{aligned} L_p = R^2 + C\sum_t \xi^t \\ - \sum_t \alpha^t \big(R^2 + \xi^t - [(\boldsymbol{x}^t)^T\boldsymbol{x}^t - 2\boldsymbol{a}^T\boldsymbol{x}^t + \boldsymbol{a}^T\boldsymbol{a}]\big) - \sum_t \gamma^t\xi^t \; . \end{aligned} \tag{13.57}$$

Dabei sind $\alpha^t \geq 0$ und $\gamma^t \geq 0$ die Lagrange-Multiplikatoren. Wenn wir die partiellen Ableitungen nach den Parametern bilden, erhalten wir

$$\frac{\partial L}{\partial R} = 2R - 2R\sum_t \alpha^t = 0 \Rightarrow \sum_t \alpha^t = 1 \tag{13.58}$$

$$\frac{\partial L}{\partial \boldsymbol{a}} = \sum_t \alpha^t(2\boldsymbol{x}^t - 2\boldsymbol{a}) = 0 \Rightarrow \sum_t \alpha^t\boldsymbol{x}^t \tag{13.59}$$

$$\frac{\partial L}{\partial \xi^t} = C - \alpha^t - \gamma^t = 0 \; . \tag{13.60}$$

Wegen $\gamma^t \geq 0$ können wir die letzte Gleichung als Nebenbedingung schreiben: $0 \leq \alpha^t \leq C$. Setzen wir dies in Gleichung 13.57 ein, erhalten wir die duale Form, die wir bezüglich α^t maximieren:

$$L_d = \sum_t \alpha^t(\boldsymbol{x}^t)^T\boldsymbol{x}^t - \sum_t\sum_s \alpha^t\alpha^s(\boldsymbol{x}^t)^T\boldsymbol{x}^s \tag{13.61}$$

unter

$$0 \leq \alpha^t \leq C \text{ und } \sum_t \alpha^t = 1 \; .$$

Wenn wir dies lösen, dann stellen wir wieder fest, dass die meisten Instanzen verschwinden und für sie $\alpha^t = 0$ ist; das sind die typischen, sehr wahrscheinlichen Instanzen, die innerhalb der Kugel liegen (Abbildung 13.10). Es gibt zwei Typen von Support-Vektoren mit $\alpha^t > 0$. Zum einen sind dies Instanzen, für die $0 < \alpha^t < C$ gilt und die auf der Grenze liegen, so dass $\|\boldsymbol{x}^t - \boldsymbol{a}\|^2 = R^2(\xi^t > 0)$; diese verwenden wir, um R zu berechnen. Instanzen, für die $\alpha^t = C(\xi^t > 0)$ gilt, liegen außerhalb der Grenze und sind Ausreißer. In Gleichung 13.59 sehen wir, dass der Mittelpunkt $\boldsymbol{a}$ als gewichtete Summe von Support-Vektoren geschrieben werden kann.

Für eine gegebene Testinstanz können wir dann sagen, dass es sich um einen Ausreißer handelt, falls

$$\|\boldsymbol{x} - \boldsymbol{a}\|^2 > R^2$$

oder

$$\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{a}^T\boldsymbol{x} + \boldsymbol{a}^T\boldsymbol{a} > R^2 \,.$$

Die Verwendung von Kernel-Funktionen erlaubt es uns, die Kugel aufzugeben und Grenzen von beliebiger Gestalt zu definieren. Indem wir das Skalarprodukt durch eine Kernel-Funktion ersetzen, erhalten wir (unter den gleichen Nebenbedingungen)

$$L_d = \sum_t \alpha^t K(\boldsymbol{x}^t, \boldsymbol{x}^t) - \sum_t \sum_s \alpha^t \alpha^s K(\boldsymbol{x}^t, \boldsymbol{x}^s) \,. \tag{13.62}$$

Beispielsweise können wir mit einem polynomialen Kernel vom Grad 2 beliebige quadratische Flächen verwenden. Mit einem Gaußschen Kernel (Gleichung 13.30) bekommen wir eine Vereinigung von lokalen Kugeln. Wir lehnen $\boldsymbol{x}$ als Ausreißer ab, wenn

$$K(\boldsymbol{x}, \boldsymbol{x}) - 2\sum_t \alpha^t K(\boldsymbol{x}, \boldsymbol{x}^t) + \sum_t \sum_s \alpha^t \alpha^s K(\boldsymbol{x}^t, \boldsymbol{x}^s) > R^2 \,.$$

Der dritte Term hängt nicht von $\boldsymbol{x}$ ab und ist daher eine Konstante (wir verwenden diese als eine Gleichung, um nach R aufzulösen, wobei $\boldsymbol{x}$ eine Instanz des Margins ist). Im Falle eines Gaußschen Kernels, für den $K(\boldsymbol{x}, \boldsymbol{x}) = 1$ gilt, reduziert sich die Bedingung auf

$$\sum_t \alpha^t K_G(\boldsymbol{x}, \boldsymbol{x}^t) < R_c$$

für die gleiche Konstante R_c. Dies ist analog zu dem Schätzer für die Kernel-Dichte (Abschnitt 8.2.2) – abgesehen davon, das die Lösung dünn besetzt ist – mit einem Schwellwert R_c für die Wahrscheinlichkeit (siehe Abbildung 13.11).

Es gibt auch einen alternativen, äquivalenten v-SVM-Typ für die Formulierung von Einklassen-SVMs, bei dem die kanonische Form der Glattheit, $(\frac{1}{2})\|\boldsymbol{w}\|^2$ verwendet wird (Schölkopf et al. 2001).

13.13 Breiter-Margin-Nächster-Nachbar-Klassifikator

In Kapitel 8 haben wir nichtparametrische Verfahren diskutiert, bei denen wir kein globales Modell an die Daten anpassen, sondern basierend auf einer Teilmenge von Nachbarinstanzen interpolieren. Speziell in Ab-

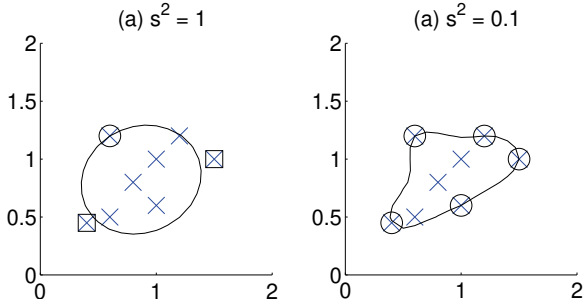


***Abb. 13.11*:** Einklassen-SVM unter Verwendung eines Gaußschen Kernels mit verschiedenen Streuungen.

schnitt 8.6 haben wir uns damit befasst, wie wichtig es ist, dabei ein gutes Abstandsmaß zu benutzen. Hier soll nun ein Verfahren vorgestellt werden, mit dem ein Abstandsmaß aus den Daten gelernt werden kann. Streng genommen handelt es sich bei diesem Verfahren nicht um eine Kernel-Maschine, sondern es basiert auf der Idee, beim Ranking einen Margin einzuhalten, was wir in Abschnitt 13.11 erörtert haben.

Die grundlegende Idee besteht darin, die k-NN-Klassifikation (siehe Abschnitt 8.4) als ein Ranking-Problem aufzufassen. Nehmen wir an, die Menge der k-nächsten Nachbarn von $\boldsymbol{x}^i$ enthält zwei Instanzen $\boldsymbol{x}^j$ und $\boldsymbol{x}^l$, wobei $\boldsymbol{x}^j$ zur gleichen Klasse gehört wie $\boldsymbol{x}^i$, während $\boldsymbol{x}^l$ zu einer anderen Klasse gehört. In einem solchen Fall sind wir einem Abstandsmaß interessiert, das für das Paar $\boldsymbol{x}^i$, $\boldsymbol{x}^l$ einen größeren Abstand liefern muss als für das Paar $\boldsymbol{x}^i$, $\boldsymbol{x}^j$. Tatsächlich fordern wir nicht einfach nur, dass er größer ist, sondern dass ein Margin von einer Einheit zwischen beiden liegt, und wenn dies nicht erfüllt ist, haben wir eine Schlupfvariable für die Differenz:

$$\mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^l) \geq \mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^j) + 1 - \xi^{ijl} \,.$$

Dieses Abstandsmaß funktioniert wie eine Score-Funktion bei einem Ranking-Problem, und jedes Tripel $(\boldsymbol{x}^i, \boldsymbol{x}^j, \boldsymbol{x}^l)$ definiert eine Rang-Nebenbedingung wie in Gleichung 13.53.

LMNN-Algorithmus

Auf dieser Idee beruht der *Breiter-Margin-Nächster-Nachbar-Algorithmus* (engl. large margin nearest neighbor, Abk. *LMNN*) (Weinberger und Saul 2009). Die zu minimierende Fehlerfunktion ist

$$(1-\mu)\sum_{i,j}\mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^j) + \mu\sum_{i,j,l}(1-y_{il})\xi_{ijl} \tag{13.63}$$

unter

$$\begin{aligned}\mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^l) &\geq \mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^j) + 1 - \xi^{ijl}, \text{ falls } \boldsymbol{r}^i = \boldsymbol{r}^j \text{ und } \boldsymbol{r}^i \neq \boldsymbol{r}^l \\ \xi^{ijl} &\geq 0 \,. \end{aligned} \tag{13.64}$$

Hierbei ist $\boldsymbol{x}^j$ einer der k-nächsten Nachbarn von $\boldsymbol{x}^i$ und sie gehören beide zur gleichen Klasse: $\boldsymbol{r}^i = \boldsymbol{r}^j$. Dieser Nachbar ist ein *Ziel.* $\boldsymbol{x}^l$ ist ebenfalls einer der k-nächsten Nachbarn von $\boldsymbol{x}^i$; falls sie die gleiche Zuordnung haben, dann wird y_{il} auf 1 gesetzt und es fällt kein Verlust an; wenn sie zu verschiedenen Klassen gehören, dann ist $\boldsymbol{x}^l$ ein *Täuscher*; y_{il} wird auf 0 gesetzt und die Bedingung 13.64 ist nicht erfüllt. Der Schlupf definiert die Kosten. Der zweite Term von Gleichung 13.63 ist die Summe dieser Schlupfe. Der erste Term ist der Gesamtabstand zu allen Zielnachbarn, und die Minimierung dieser Größe hat den Effekt einer Regularisierung – wir wollen die Abstände so klein wie möglich halten.

Bei LMNN wird die Mahalanobis-Distanz als Abstandsmaß verwendet:

$$\mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^j|\mathbf{M}) = (\boldsymbol{x}^i - \boldsymbol{x}^j)^T\mathbf{M}(\boldsymbol{x}^i - \boldsymbol{x}^j)\,. \tag{13.65}$$

Die Matrix $\mathbf{M}$ ist dabei der zu optimierende Parameter. Gleichung 13.63 definiert ein konvexes (genauer gesagt: positiv semi-definites) Problem und hat folglich ein eindeutiges Minimum.

Wenn die Eingabedimensionalität hoch ist und es nur wenige Daten gibt, was wir im Zusammenhang mit Gleichung 8.21 diskutiert hatten, dann können wir regularisieren, indem wir $\mathbf{M}$ in $\mathbf{L}^T\mathbf{L}$ faktorisieren, wobei $\mathbf{L}$ eine $k \times d$-Matrix mit $k < d$ ist:

$$\mathcal{D}(\boldsymbol{x}^i, \boldsymbol{x}^j|\mathbf{L}) = \|\mathbf{L}\boldsymbol{x}^i - \mathbf{L}\boldsymbol{x}^j\|^2\,. \tag{13.66}$$

$\mathbf{L}\boldsymbol{x}$ ist die k-dimensionale Projektion von $\boldsymbol{x}$, und die Mahalanobis-Distanz im originalen, d-dimensionalen $\boldsymbol{x}$-Raum entspricht der (quadrierten) Euklidischen Distanz im neuen, k-dimensionalen Raum; ein Beispiel ist in Abbildung 8.6 zu sehen. Wenn wir Gleichung 13.66 als Abstandsmaß in Gleichung 13.63 einsetzen, dann erhalten wir den *LMCA-Algorithmus* (Komponentenanalyse mit breitem Margin, engl. *large margin component analysis;* Torresani und Lee 2007). Das Problem ist leider kein konvexes Optimierungsproblem mehr, und wenn wir erhalten mittels Gradientenabstieg lokal optimale Lösungen.

LMCA-Algorithmus

13.14 Dimensionalitätsreduktion mit Kernel

Wir wissen aus Abschnitt 6.3, dass die Hauptkomponentenanalyse (PCA) die Dimensionalität reduziert, indem auf diejenigen Eigenvektoren der Kovarianzmatrix $\boldsymbol{\Sigma}$ projiziert wird, die die größten Eigenwerte haben. Wenn die Dateninstanzen zentriert sind ($E[\boldsymbol{x}] = 0$), kann diese Projektion in der Form $\mathbf{X}^T\mathbf{X}$ geschrieben werden. In der Kernel-Version arbeiten wir in dem Raum der $\boldsymbol{\phi}(\boldsymbol{x})$ anstatt im originalen $\boldsymbol{x}$-Raum, und da die

Dimensionalität D dieses neuen Raums wie gewöhnlich größer sein kann als die Größe N der Datenmenge, ziehen wir es vor, mit der $N \times N$-Matrix $\mathbf{X}\mathbf{X}^T$ zu arbeiten und eine Merkmalseinbettung durchzuführen, anstatt mit der $d \times d$-Matrix $\mathbf{X}^T\mathbf{X}$ zu arbeiten. Die projizierte Datenmatrix ist $\mathbf{\Phi} = \boldsymbol{\phi}(\mathbf{X})$, folglich arbeiten wir mit den Eigenvektoren von $\mathbf{\Phi}^T\mathbf{\Phi}$, d. h. mit der Kernel-Matrix $\mathbf{K}$.

KERNEL-PCA

Kernel-PCA verwendet die Eigenvektoren und Eigenwerte der Kernel-Matrix, was einer linearen Dimensionalitätsreduktion im $\phi(\boldsymbol{x})$-Raum entspricht. Wenn die $\boldsymbol{c}_i$ und λ_i die zugehörigen Eigenvektoren und Eigenwerte sind, dann können die neuen, k-dimensionale Werte wie folgt berechnet werden:

$$z_j^t = \sqrt{\lambda_j} c_j^t, \; j = 1, \ldots, k, \; t = 1, \ldots, N \; .$$

Ein Beispiel ist in Abbildung 13.12 zu sehen, wo wir zuerst einen quadratischen Kernel benutzen, dann mit einem PCA-Kernel die Dimensionalität auf zwei (von fünf) reduzieren und schließlich eine lineare SVM implementieren. Man beachte, dass im allgemeinen Fall (etwa mit einem Gauß-Kernel) die Eigenwerte nicht notwendigerweise fallen und dass es keine Garantie dafür gibt, dass wir die Dimensionalität mit einem PCA-Kernel tatsächlich reduzieren können.

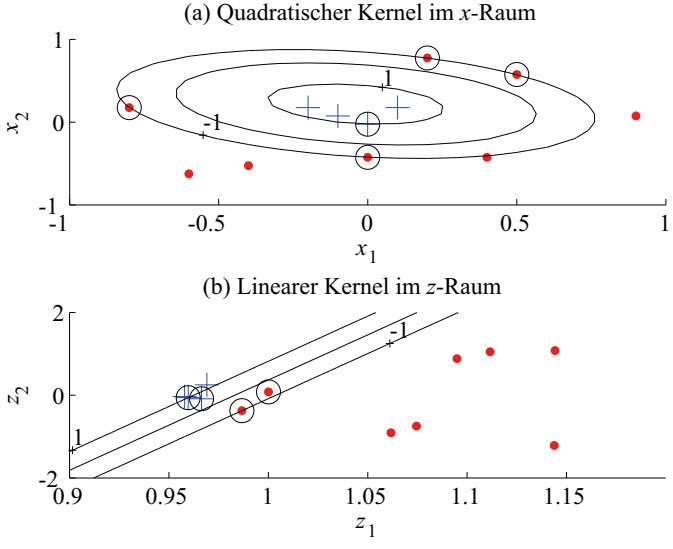


Abb. 13.12: Anstatt einen quadratischen Kernel im Originalraum zu verwenden, können wir (a) einen PCA-Kernel auf die quadratischen Kernel-Werte anwenden, um in einen zweidimensionalen neuen Raum abzubilden, in dem wir (b) eine lineare Diskriminanzfunktion verwenden können. Diese zwei Dimensionen (von fünf) erklären 80 Prozent der Varianz.

Was wir hier tun, ist multidimensionales Skalieren (Abschnitt 6.7), wobei wir Kernel-Werte als Ähnlichkeitswerte verwenden. Wenn wir beispielsweise $k = 2$ setzen, können wir die Daten in dem durch die Kernel-Matrix induzierten Raum visualisieren. Dies kann uns Hinweise darauf liefern, wie der verwendete Kernel Ähnlichkeit definiert. Für die lineare Diskriminanzanalyse (LDA) (Abschnitt 6.8) kann auf ähnliche Weise eine Kernel-Version konstruiert werden (Müller et al. 2001). Die Kernel-Version der kanonischen Korrelationsanalyse (CCA) (Abschnitt 6.9) wird in Hardoon, Szedmak, Shawe-Taylor 2004 diskutiert.

In Kapitel 6 haben wir uns mit Verfahren zur nichtlinearen Dimensionalitätsreduktion, Isomap und LLE beschäftigt. Tatsächlich können wir, wenn wir die Elemente der Kostenmatrix in Gleichung 6.58 als Kernel-Auswertungen für Paare von Eingaben auffassen, LLE als Kernel-PCA für eine spezielle Wahl des Kernels interpretieren. Das gleiche gilt für Isomap, wenn eine Kernel-Funktion als Funktion des geodätischen Abstands auf dem Graphen definiert wird.

13.15 Anmerkungen

Lineare Modelle zu verallgemeinern, indem die Daten mithilfe nichtlinearer Basisfunktionen in einen neuen Raum abgebildet werden, ist keine neue Idee. Die Neuartigkeit von Support-Vektor-Maschinen besteht darin, diese Idee in einen Lernalgorithmus zu integrieren, dessen Parameter durch eine Teilmenge von Dateninstanzen (die sogenannte *duale Darstellung*) definiert sind. Es ist folglich auch nicht notwendig, die Basisfunktionen explizit auszuwerten, wodurch auch die Komplexität aufgrund der Größe der Trainingsmenge beschränkt wird. Dies gilt auch für Gaußsche Prozesse, bei denen die Kernel-Funktion Kovarianzmatrix genannt wird (Abschnitt 16.9).

DUALE DARSTELLUNG

Die Tatsache, dass die Lösung dünn besetzt ist, zeigt den Vorteil gegenüber nichtparametrischen Schätzern wie k-NN und Parzen-Fenstern, und die Flexibilität bei der Verwendung von Kernel-Funktionen ermöglicht es, mit nichtvektoriellen Daten zu arbeiten. Da es eine eindeutige Lösung für das Optimierungsproblem gibt, müssen wir keine iterative Optimierungsprozedur ausführen wie bei neuronalen Netzen. Aus all diesen Gründen werden Support-Vektor-Maschinen heute als die besten Lerner „von der Stange“ angesehen. Daher haben sie große Verbreitung in vielen Anwendungsgebieten gefunden, besonders aber in der Bioinformatik (Schölkopf, Tsuda und Vert 2004) und bei der natürlichen Sprachverarbeitung, wobei eine wachsende Zahl von Tricks für das Herleiten von Kerneln zum Einsatz kommt (Shawe-Taylor und Cristianini 2004).

Die Verwendung von Kernel-Funktionen bringt eine andere Darstellung der Daten mit sich. Wir definieren eine Instanz nicht mehr direkt als einen Vektor von Attributen, sondern dadurch, wie ähnlich sie anderen

Instanzen ist bzw. wie stark sie sich von ihnen unterscheidet. Dies ähnelt dem Unterschied zwischen der multidimensionalen Skalierung, bei der eine Matrix von Abständen verwendet wird (wobei es hier nicht nötig ist zu wissen, wie diese berechnet wurden) und der Hauptkomponentenanalyse, bei der Vektoren irgendeines Raumes verwendet werden.

Die Support-Vektor-Maschine gilt heute als der beste „gebrauchsfertige" Lernalgorithmus und sie wird in vielen Gebieten erfolgreich eingesetzt. Durch die Tatsache, dass wir ein konvexes Problem lösen und daher eine optimale Lösung bekommen sowie die Idee der Kernels, die es uns erlauben, a-priori-Informationen zu codieren, ist das Verfahren sehr populär geworden. Es gibt eine sehr unfangreiche Literatur über Support-Vektor-Maschinen und allgemein für Kernel-Maschinen. Die klassischen Bücher sind Vapnik (1995, 1998) sowie Schölkopf und Smola (2002). Burges (1998) sowie Smola und Schölkopf (1998) sind gute Tutorials für den Einsatz von SVMs bei der Klassifikation bzw. Regression. Zudem sind viele freie Software-Pakete erhältlich, von denen SVMlight (Joachims 2008) und LIBSVM (Chang und Lin 2011) die populärsten sind.

13.16 Übungen

1. Schlagen Sie einen Filteralgorithmus vor, mit dem Trainingsinstanzen gefunden werden können, die mit sehr geringer Wahrscheinlichkeit Support-Vektoren sind.

   Lösung: Support-Vektoren sind diejenigen Instanzen, die nahe an den Grenzen liegen. Eine Instanz, die von vielen Instanzen umgeben ist, die alle zur gleichen Klasse gehören, wird also sehr wahrscheinlich nicht als Support-Vektor ausgewählt werden. Beispielsweise können wir eine 11-Nächste-Nachbarn-Suche für alle Instanzen durchführen, und wenn alle 11 Nachbarn zur selben Klasse gehören, entfernen wir diese Instanzen aus der Trainingsmenge.

2. Wie können wir in Gleichung 13.31 $\mathbf{S}$ schätzen?

   Lösung: Wir können die Kovarianzmatrix der Daten berechnen und diese als $\mathbf{S}$ verwenden. Eine andere Möglichkeit ist, für jeden Support-Vektor ein lokales $\mathbf{S}^t$ zu haben und diesen durch eine Reihe von benachbarten Datenpunkten zu schätzen. In diesem Fall kann es sein, dass wir Maßnahmen ergreifen müssen, um sicherzustellen, dass $\mathbf{S}$ nicht singulär ist, oder dass wir die Dimensionalität verringern müssen.

3. Wie können wir bei der empirischen Kernel-Map die Templates wählen?

   Lösung: Der einfachste und am häufigsten verwendete Ansatz besteht darin, alle Trainingsinstanzen zu verwenden. In diesem Fall ist $\boldsymbol{\phi}(\cdot)$ N-dimensional. Wir können die Komplexität verringern und

das Modell effizienter machen, indem wir eine Teilmenge auswählen. Diese Auswahl kann zufällig erfolgen oder durch eine Clusteranalyse, wobei die Clusterzentren als Templates verwendet werden (wie bei der Vektorquantisierung), oder wir verwenden eine Teilmenge, die den Eingaberaum gut abdeckt und dabei möglichst wenige Instanzen umfasst.

4. Schlagen Sie für die lokalisierte Multi-Kernel-Gleichung 13.40 ein geeignetes Modell für $\eta_i(\boldsymbol{x}|\theta_i)$ vor und erläutern Sie, wie dieses trainiert werden kann.

5. Welche Beziehung gibt es bei der Kernel-Regression zwischen ϵ und der Varianz des Rauschens?

6. Welchen Einfluss hat ϵ bei der Kernel-Regression auf die Verzerrung und die Varianz?

   Lösung: ϵ ist ein Glättungsparameter. Wenn dieser zu groß ist, dann glätten wir zu stark, d. h., wir reduzieren die Varianz, riskieren aber ein Anwachsen der Verzerrung. Ist die Glättung zu klein, haben wir eine große Varianz und die Verzerrung wird klein.

7. Leiten Sie die Kernel-Version der primalen und der dualen Form sowie der Bewertungsfunktion für das Ranking-Problem her.

   Lösung: Die primale Form ist

   $$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_t \xi^t$$

   unter

   $$\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}^u - \boldsymbol{x}^v) \geq 1 - \xi^t\,,$$
   $$\xi^t \geq 0\,.$$

   Die duale Form ist

   $$L_d = \sum_t \alpha^t - \frac{1}{2}\sum_t\sum_s \alpha^t\alpha^s K(\boldsymbol{x}^u - \boldsymbol{x}^v, \boldsymbol{x}^k - \boldsymbol{x}^l)$$

   mit $K(\boldsymbol{x}^u - \boldsymbol{x}^v, \boldsymbol{x}^k - \boldsymbol{x}^l) = \boldsymbol{\phi}(\boldsymbol{x}^u - \boldsymbol{x}^v)^T\boldsymbol{\phi}(\boldsymbol{x}^k - \boldsymbol{x}^l)$.

   Für eine neue Testinstanz $\boldsymbol{x}$ wird die Bewertung wie folgt berechnet:

   $$g(\boldsymbol{x}) = \sum_t \alpha^t K(\boldsymbol{x}^u - \boldsymbol{x}^v, \boldsymbol{x})\,.$$

8. Wie können Einklassen-SVMs für die Klassifikation verwendet werden?

   Lösung: Wir können für jede Klasse eine separate Einklassen-SVM verwenden und diese dann kombinieren, um eine Entscheidung zu

treffen. Beispielsweise können wir für jede Klasse C_i eine Einklassen-SVM anpassen, um die Parameter α_i^t zu finden:

$$\sum_t \alpha_i^t K_G(\boldsymbol{x}, \boldsymbol{x}^t) \,.$$

Dies kann dann als Schätzer für $p(\boldsymbol{x}|C_i)$ genommen werden. Wenn die a-priori-Wahrscheinlichkeiten mehr oder weniger gleich sind, können wir einfach die Klasse mit dem höchsten Wert wählen; anderenfalls können wir den Satz von Bayes für die Klassifikation benutzen.

9. Wenden Sie in einer Konstellation wie in Abbildung 13.12 Kernel-PCA mit einem Gauß-Kernel an.

10. Angenommen, wir haben zwei Repräsentationen des gleichen Objekts und dazu zwei verschiedene Kernel. Wie können wir unter Verwendung dieser beiden Kernel eine Dimensionalitätsreduktion mit Kernel-PCA implementieren?

13.17 Literaturangaben

Allwein, E. L., R. E. Schapire, and Y. Singer. 2000. „Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers.“ *Journal of Machine Learning Research* 1: 113–141.

Burges, C. J. C. 1998. „A Tutorial on Support Vector Machines for Pattern Recognition.“ *Data Mining and Knowledge Discovery* 2: 121–167.

Chang, C.-C., and C.-J. Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology* 2: 27: 1–27:27.

Chapelle, O., and S. S. Keerthi. 2010. „Efficient Algorithms for Ranking with SVMs.“ *Information Retrieval* 11: 201–215.

Cherkassky, V., and F. Mulier. 1998. *Learning from Data: Concepts, Theory, and Methods.* New York: Wiley.

Cortes, C., and V. Vapnik. 1995. „Support Vector Networks.“ *Machine Learning* 20: 273–297.

Dietterich, T. G., and G. Bakiri. 1995. „Solving Multiclass Learning Problems via Error-Correcting Output Codes.“ *Journal of Artificial Intelligence Research* 2: 263–286.

Gönen, M., and E. Alpaydın. 2008. „Localized Multiple Kernel Learning.“ In *25th International Conference on Machine Learning,* ed. A. McCallum and S. Roweis, 352–359. Madison, WI: Omnipress.

Gönen, M., and E. Alpaydın. 2011. „Multiple Kernel Learning Algorithms.“ *Journal of Machine Learning Research* 12: 2211–2268.

Hardoon, D. R., S. Szedmak, J. Shawe-Taylor. 2004. „Canonical Correlation Analysis: An Overview with Application to Learning Methods.“ *Neural Computation* 16: 2639–2664.

Herbrich, R., K. Obermayer, and T. Graepel. 2000. „Large Margin Rank Boundaries for Ordinal Regression.“ In *Advances in Large Margin Classifiers,* ed. A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, 115–132. Cambridge, MA: MIT Press.

Jaakkola, T., and D. Haussler. 1999. „Exploiting Generative Models in Discriminative Classifiers.“ In *Advances in Neural Information Processing Systems 11,* ed. M. J. Kearns, S. A. Solla, and D. A. Cohn, 487–493. Cambridge, MA: MIT Press.

Joachims, T. 2002. „Optimizing Search Engines using Clickthrough Data.“ In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 133–142. New York, NY: ACM.

Joachims, T. 2008. SVMlight, `http://svmlight.joachims.org`.

Lanckriet, G. R., N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. 2004. „Learning the Kernel Matrix with Semidefinite Programming.“ *Journal of Machine Learning Research* 5: 27–72.

Liu, T.-Y. 2011. *Learning to Rank for Information Retrieval.* Heidelberg: Springer.

Mayoraz, E., and E. Alpaydın. 1999. „Support Vector Machines for Multiclass Classification.“ In *Foundations and Tools for Neural Modeling, Proceedings of IWANN'99, LNCS 1606,* ed. J. Mira and J. V. Sanchez-Andres, 833–842. Berlin: Springer.

Müller, K. R., S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. 2001. „An Introduction to Kernel-Based Learning Algorithms.“ *IEEE Transactions on Neural Networks* 12: 181–201.

Noble, W. S. 2004. „Support Vector Machine Applications in Computational Biology.“ In *Kernel Methods in Computational Biology,* ed. B. Schölkopf, K. Tsuda, and J.-P. Vert, 71–92. Cambridge, MA: MIT Press.

Platt, J. 1999. „Probabilities for Support Vector Machines.“ In *Advances in Large Margin Classifiers,* ed. A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, 61–74. Cambridge, MA: MIT Press.

Schölkopf, B., J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. 2001. „Estimating the Support of a High-Dimensional Distribution.“ *Neural Computation* 13: 1443–1471.

Schölkopf, B., and A. J. Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA: MIT Press.

Schölkopf, B., A. J. Smola, R. C. Williamson, and P. L. Bartlett. 2000. „New Support Vector Algorithms." *Neural Computation* 12: 1207–1245.

Schölkopf, B., K. Tsuda, and J.-P. Vert, eds. 2004. *Kernel Methods in Computational Biology.* Cambridge, MA: MIT Press.

Shawe-Taylor, J., and N. Cristianini. 2004. *Kernel Methods for Pattern Analysis.* Cambridge, UK: Cambridge University Press.

Smola, A., and B. Schölkopf. 1998. *A Tutorial on Support Vector Regression,* NeuroCOLT TR-1998-030, Royal Holloway College, University of London, UK.

Sonnenburg, S., G. Rätsch, C. Schäfer, and B. Schölkopf. 2006. „Large Scale Multiple Kernel Learning." *Journal of Machine Learning Research* 7: 1531– 1565.

Tax, D. M. J., and R. P. W. Duin. 1999. „Support Vector Domain Description." *Pattern Recognition Letters* 20: 1191–1199.

Torresani, L., and K. C. Lee. 2007. „Large Margin Component Analysis." In *Advances in Neural Information Processing Systems* 19, ed. B. Schölkopf, J. Platt, and T. Hoffman, 1385–1392. Cambridge, MA: MIT Press.

Vapnik, V. 1995. *The Nature of Statistical Learning Theory.* New York: Springer.

Vapnik, V. 1998. *Statistical Learning Theory.* New York: Wiley.

Vert, J.-P., K. Tsuda, and B. Schölkopf. 2004. „A Primer on Kernel Methods." In *Kernel Methods in Computational Biology,* ed. B. Schölkopf, K. Tsuda, and J.-P. Vert, 35–70. Cambridge, MA: MIT Press.

Weinberger, K. Q., and L. K. Saul. 2009. „Distance Metric Learning for Large Margin Classification." *Journal of Machine Learning Research* 10: 207–244.

Weston, J., and C. Watkins. 1998. „Multiclass Support Vector Machines." *Technical Report CSD-TR-98-04,* Department of Computer Science, Royal Holloway, University of London.

14 Graphenmodelle

Graphenmodelle stellen die Wechselwirkungen zwischen Variablen dar und haben den Vorteil, dass die Deduktion über eine große Anzahl von Variablen durch Ausnutzen bedingter Unabhängigkeiten in eine Menge von lokalen Berechnungen zerlegt werden kann, bei denen jeweils nur eine kleine Zahl von Variablen beteiligt ist. Nachdem wir uns einige Beispiele für Inferenzen von Hand angesehen haben, diskutieren wir das Konzept der d-Separation und den Belief-Propagation-Algorithmus auf einer Reihe von Graphen.

14.1 Einführung

GRAPHENMODELLE
BAYESSCHE NETZE
BELIEF-NETZE
PROBABILISTISCHE NETZE

Graphenmodelle, auch *Bayessche Netze*, *Belief-Netze* oder *probabilistische Netze* genannt, bestehen aus Knoten und sie verbindenden Kanten. Jeder Knoten entspricht einer Zufallsvariable X, und er hat einen Wert, welcher der Wahrscheinlichkeit der Zufallsvariable, $P(X)$, entspricht. Wenn es eine gerichtete Kante vom Knoten X zum Knoten Y gibt, dann bedeutet das, dass X einen direkten Einfluss auf Y hat. Dieser Einfluss wird durch die bedingte Wahrscheinlichkeit $P(Y|X)$ spezifiziert. Das Netz ist ein *gerichteter azyklischer Graph* (DAG für engl. *directed acyclic graph*), d. h., es gibt keine Zyklen. Die Knoten und die Kanten zwischen ihnen definieren die *Struktur* des Netzes, und die bedingten Wahrscheinlichkeiten sind die *Parameter* bei gegebener Struktur.

GERICHTETER AZYKLISCHER GRAPH

Ein einfaches Beispiel ist in Abbildung 14.1 zu sehen: Es modelliert, dass Regen nasses Gras verursacht. Es regnet an 40 % aller Tage, und wenn es regnet, dann gibt es eine 90-prozentige Chance, dass das Gras nass wird. An 10 % der Tage regnet es möglicherweise nicht lange genug, als dass wir das Gras wirklich als nass einstufen könnten. Die Zufallsvariablen in diesem Beispiel sind binär: wahr oder falsch. Es gibt eine Wahrscheinlichkeit von 20 %, dass das Gras nass wird, ohne dass es tatsächlich regnet, beispielsweise wenn eine Sprinkleranlage genutzt wird.

Wie wir sehen, ist durch diese drei Wahrscheinlichkeiten die gemeinsame Verteilung von $P(R, W)$ vollständig spezifiziert. Wenn $P(R) = 0{,}4$, dann ist $P(\sim R) = 0{,}6$ und entsprechend ist $P(\sim W|R) = 0{,}1$ und $P(\sim W|\sim R) = 0{,}8$. Die Verbundwahrscheinlichkeit ist

$$P(R, W) = P(R)\, P(W|R)\,.$$

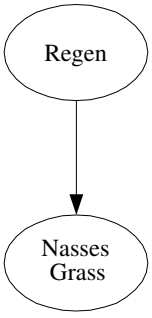


Abb. 14.1: Bayessches Netz, welches Regen als Ursache für nasses Gras modelliert.

Wir können die individuelle (marginale) Wahrscheinlichkeit von nassem Gras berechnen, indem wir über die möglichen Werte summieren, welche der Elternknoten annehmen kann:

$$\begin{aligned} P(W) &= \sum_R P(R,W) = P(W|R)\,P(R) + P(W|\sim R)\,P(\sim R) \\ &= 0{,}9 \cdot 0{,}4 + 0{,}2 \cdot 0{,}6 = 0{,}48\,. \end{aligned}$$

Wenn wir wüssten, dass es geregnet hat, wäre die Wahrscheinlichkeit für nasses Gras 0,9; wenn wir mit Sicherheit wüssten, dass es nicht geregnet hat, wäre sie nur 0,2; und wenn wir nicht sicher wären, ob es geregnet hat, wäre sie 0,48.

KAUSALER GRAPH

Abbildung 14.1 zeigt einen *kausalen Graphen*: Er erklärt, dass die Ursache für nasses Gras Regen ist. Der Satz von Bayes erlaubt uns, die Abhängigkeiten umzukehren und eine *Diagnose* vorzunehmen. Wenn wir zum Beispiel wissen, dass das Gras nass ist, kann die Wahrscheinlichkeit dafür, dass es geregnet hat, wie folgt berechnet werden:

$$P(R|W) = \frac{P(W|R)\,P(R)}{P(W)} = 0{,}75\,.$$

Das Wissen, dass das Gras nass ist, hat die Wahrscheinlichkeit von Regen von 0,4 auf 0,75 erhöht; das liegt daran, dass $P(W|R)$ groß und $P(W|\sim R)$ klein ist.

UNABHÄNGIGKEIT

Wir bilden Graphen, indem wir Knoten und Kanten hinzufügen und dabei Abhängigkeiten generieren. X und Y sind *unabhängige Ereignisse*, wenn

$$P(X,Y) = P(X)\,P(Y)\,. \tag{14.1}$$

X und Y sind *bedingt unabhängige Ereignisse* gegeben ein drittes Ereignis Z, wenn

$$P(X,Y|Z) = P(X|Z)\,P(Y|Z)\,, \tag{14.2}$$

was auch in der Form

$$P(X|Y,Z) = P(X|Z) \tag{14.3}$$

geschrieben werden kann.

In einem Graphenmodell sind nicht alle Knoten verbunden; tatsächlich ist ein Knoten im Allgemeinen nur mit einer sehr kleinen Zahl anderer Knoten verbunden. Bestimmte Teilgraphen beinhalten Aussagen über bedingte Unabhängigkeiten, und diese erlauben es uns, einen komplexen Graphen in kleinere Teilmengen zu zerlegen, in denen Inferenzen lokal abgeleitet werden können und deren Ergebnisse später durch den Graphen propagieren. Es gibt drei kanonische Fälle, aus denen größere Graphen zusammengesetzt werden können.

14.2 Kanonische Fälle für bedingte Unabhängigkeit

Fall 1: Spitze-Ende-Verbindung

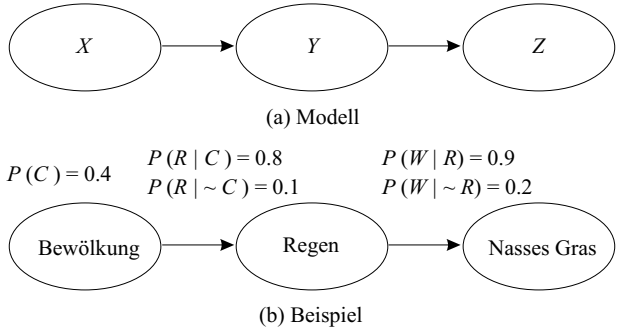


Abb. 14.2: Spitze-Ende-Verbindung. (a) Drei Knoten sind seriell verbunden. X und Z sind unabhängig gegeben den Zwischenknoten Y: $P(Z|Y,X) = P(Z|Y)$. (b) Beispiel: Bewölkung verursacht Regen und dieser wiederum nasses Gras.

Drei Ereignisse können wie in Abbildung 14.2a seriell verbunden sein. Wie wir hier sehen, sind X und Z unabhängig gegeben Y: Das Wissen über Y sagt Z alles, die Kenntnis des Zustands von X fügt zu Z kein zusätzliches Wissen hinzu. Wir schreiben dies als $P(Z|Y,X) = P(Z|Y)$. Wir sagen, dass Y den Weg von X nach Z *blockiert* oder anders formuliert, Y trennt sie in dem Sinne, dass die Entfernung von Y dazu führt, dass es keinen Weg zwischen X und Z gibt. In diesem Fall kann die Verbundwahrscheinlichkeit geschrieben werden als

$$P(X,Y,Z) = P(X)\,P(Y|X)\,P(Z|Y)\,. \tag{14.4}$$

Die Verbundwahrscheinlichkeit in dieser Weise zu schreiben, bedeutet Unabhängigkeit:

$$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(X)P(Y|X)P(Z|Y)}{P(X)P(Y|X)} = P(Z|Y)\,. \qquad (14.5)$$

Typischerweise ist X die Ursache von Y und Y ist ist die Ursache von Z. In dem Beispiel in Abbildung 14.2b steht X für Bewölkung, Y für Regen und Z für nasses Gras. Wir können Information über diese Kette propagieren lassen. Wenn wir den Bewölkungszustand nicht kennen, haben wir

$$P(R) = P(R|C)\,P(C) + P(R|\sim C)\,P(\sim C) = 0{,}38\,,$$
$$P(W) = P(W|R)\,P(R) + P(W|\sim R)\,P(\sim R) = 0{,}48\,.$$

Nehmen wir nun an, wir stellen am Morgen fest, dass es bewölkt ist. Was können wir über die Wahrscheinlichkeit aussagen, dass das Gras nass wird? Um das zu beantworten, muss die Evidenz zuerst über den Zwischenknoten R und dann zum Abfrageknoten W propagieren:

$$P(W|C) = P(W|R)\,P(R|C) + P(W|\sim R)\,P(\sim R|C) = 0{,}76\,.$$

Das Wissen, dass es bewölkt ist, hat die Wahrscheinlichkeit von nassem Gras erhöht. Mit dem Satz von Bayes können wir die Evidenz auch zurückverfolgen. Angenommen, wir sind morgens weggefahren und stellen abends beim Nachhausekommen fest, dass unser Rasen nass ist. Wie groß ist die Wahrscheinlichkeit, dass es an diesem Tag bewölkt war? Wir verwenden den Satz von Bayes und kehren die Richtung um:

$$P(C|W) = \frac{P(W|C)\,P(C)}{P(W)} = 0{,}65\,.$$

Das Wissen, dass das Gras nass ist, hat die Wahrscheinlichkeit für bewölktes Wetter von ihrem Defaultwert (dem a-priori-Wert) 0,4 auf 0,65 erhöht.

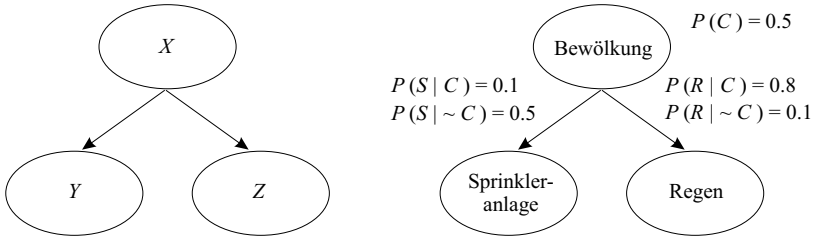
Fall 2: Ende-Ende-Verbindung

X kann, wie in Abbildung 14.3a skizziert, der Elternknoten von zwei Knoten X und Z sein. Die Verbunddichte lässt sich schreiben als

$$P(X,Y,Z) = P(X)\,P(Y|X)\,P(Z|X)\,. \qquad (14.6)$$

Normalerweise sind Y und Z vermittel durch X abhängig, doch wenn X gegeben ist, werden sie unabhängig:

$$P(Y,Z|X) = \frac{P(X,Y,Z)}{P(X)} = \frac{P(X)\,P(Y|X)\,P(Z|X)}{P(X)}$$
$$= P(Y|X)\,P(Z|X)\,. \qquad (14.7)$$



(a) Modell (b) Beispiel

***Abb. 14.3*:** Ende-Ende-Verbindung. X ist der Elternknoten der beiden Knoten Y und Z. Die beiden Kindknoten sind sind unabhängig gegeben X: $P(Y|X,Z) = P(Y|Z)$. In diesem Beispiel verursacht Bewölkung Regen und macht es außerdem weniger wahrscheinlich, dass wir die Sprinkleranlage anmachen.

Wenn der Wert von X bekannt ist, blockiert X den Weg zwischen Y und Z oder anders formuliert: Es trennt die beiden anderen Knoten.

In Abbildung 14.3b sehen wir ein Beispiel, in dem die Bewölkung Einfluss sowohl auf den Regen als auch auf den Einsatz der Sprinkleranlage hat (im ersten Fall ist der Einfluss positiv, im zweiten negativ). Wenn wir zum Beispiel wissen, dass es geregnet hat, können wir die Abhängigkeit mithilfe des Bayesschen Satzes umkehren und auf die Ursache schließen:

$$\begin{aligned} P(C|R) &= \frac{P(R|C)\,P(C)}{P(R)} = \frac{P(R|C)\,P(C)}{\sum_C P(R,C)} \\ &= \frac{P(R|C)\,P(C)}{P(R|C)\,P(C) + P(R|\sim C)\,P(\sim C)} = 0{,}89\,. \end{aligned} \tag{14.8}$$

Man beachte, dass dieser Wert größer ist als $P(C)$; das Wissen, dass es geregnet hat, hat die Wahrscheinlichkeit für Bewölkung erhöht.

Wenn das X in Abbildung 14.3a nicht bekannt ist, können wir zum Beispiel aus dem Wissen über Y Rückschlüsse auf X ziehen, die wir dann verwenden, um auf Z zu schließen. In Abbildung 14.3b hat das Wissen über den Zustand des Sprinklers Einfluss auf die Wahrscheinlichkeit, dass es geregnet hat. Wenn wir wissen, dass der Sprinkler an ist, gilt

$$\begin{aligned} P(R|S) &= \sum_C P(R,C|S) = P(R|C)P(C|S) + P(R|\sim C)P(\sim C|S) \\ &= P(R|C)\,\frac{P(S|C)\,P(C)}{P(S)} + P(R|\sim C)\,\frac{P(S|\sim C)\,P(\sim C)}{P(S)} \\ &= 0{,}22\,. \end{aligned} \tag{14.9}$$

Das ist kleiner als $P(R) = 0{,}45$ – das Wissen, dass die Sprinkleranlage an ist, verringert also die Wahrscheinlichkeit, dass es geregnet hat, da die Zustände „Benutzen der Sprinkleranlage“ und „Regen“ bei unterschiedlichen Zuständen der Bewölkung vorkommen. Wenn wir wissen, dass die Sprinkleranlage aus ist, erhalten wir mit dem gleichen Ansatz $P(R| \sim S) = 0{,}55$; die Wahrscheinlichkeit von Regen nimmt in diesem Fall zu.

Fall 3: Spitze-an-Spitze-Verbindung

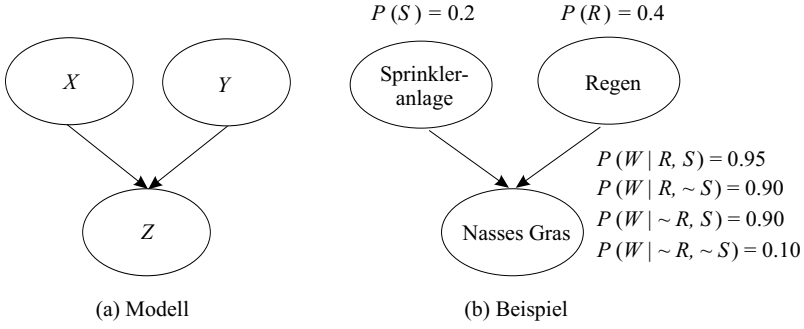


Abb. 14.4: Spitze-an-Spitze-Verbindung. Ein Knoten hat zwei Eltern, die unabhängig sind, solange das Kind nicht gegeben ist. Beispielsweise kann ein Ereignis zwei unabhängige Ursachen haben.

Bei einer Spitze-an-Spitze-Verbindung gibt es zwei Eltern, X und Y, für einen einzelnen Knoten Z (siehe Abbildung 14.4a). Die Verbunddichte lautet

$$P(X, Y, Z) = P(X)\, P(Y)\, P(Z|X, Y)\,. \tag{14.10}$$

X und Y sind unabhängig, das heißt, es gilt $P(X, Y) = P(X) \cdot P(Y)$ (Übung 2), doch wenn Z bekannt ist, werden sie abhängig. Das Konzept des Blockierens oder Trennens ist in diesem Fall anders: Der Weg zwischen X und Y ist blockiert (oder anders formuliert: die Knoten sind getrennt), wenn Z *nicht* beobachtet wird. Wenn Z (oder irgendeiner seiner Nachkommen) beobachtet wird, sind sie nicht blockiert (bzw. getrennt oder unabhängig).

In Abbildung 14.4b sehen wir zum Beispiel, dass der Knoten W zwei Eltern R und S hat, und somit ist seine Wahrscheinlichkeit bedingt durch die Werte dieser beiden, $P(W|R, S)$.

Wenn nicht anderes bekannt ist, wird die Wahrscheinlichkeit, dass das Gras nass ist, durch Marginalisieren über die Verbundwahrscheinlichkeit berechnet:

$$\begin{aligned}
P(W) &= \sum_{R,S} P(W,R,S) \\
&= P(W|R,S)\,P(R,S) + P(W|\sim R,S)\,P(\sim R,S) \\
&\quad + P(W|R,\sim S)\,P(R,\sim S) + P(W|\sim R,\sim S)\,P(\sim R,\sim S) \\
&= P(W|R,S)\,P(R)\,P(S) + P(W|\sim R,S)\,P(\sim R)\,P(S) \\
&\quad + P(W|R,\sim S)\,P(R)\,P(\sim S) + P(W|\sim R,\sim S)\,P(\sim R)\,P(\sim S) \\
&= 0{,}52\,.
\end{aligned}$$

Nehmen wir nun an, wir wissen, dass der Sprinkler an ist, und wir prüfen, wie dies die Wahrscheinlichkeit beeinflusst. Dies ist eine kausale Inferenz (Vorhersage):

$$\begin{aligned}
P(W|S) &= \sum_{R} P(W,R|S) \\
&= P(W|R,S)\,P(R|S) + P(W|\sim R,S)\,P(\sim R|S) \\
&= P(W|R,S)\,P(R) + P(W|\sim R,S)\,P(\sim R) \\
&= 0{,}92\,.
\end{aligned}$$

Wie wir sehen, ist $P(W|S) > P(W)$; wegen des Wissen, dass die Sprinkleranlage an ist, wächst die Wahrscheinlichkeit für nasses Gras.

Wir können auch die Wahrscheinlichkeit berechnen, dass die Sprinkleranlage an ist, gegeben, dass das Gras nass ist. Das ist eine diagnostische Inferenz:

$$P(S|W) = \frac{P(W|S)\,P(S)}{P(W)} = 0{,}35\,.$$

Es gilt $P(S|W) > P(S)$, d. h., das Wissen, dass das Gras nass ist erhöht die Wahrscheinlichkeit, dass der Sprinkler an ist. Nehmen wir nun an, dass es geregnet hat. Wir haben dann

$$\begin{aligned}
P(S|R,W) &= \frac{P(W|R,S)\,P(S|R)}{P(W|R)} = \frac{P(W|R,S)\,P(S)}{P(W|R)} \\
&= 0{,}21\,,
\end{aligned}$$

was kleiner ist als $P(S|W)$. Das wird als *Wegerklären* bezeichnet. Wenn wir wissen, dass es geregnet hat, verringert sich die Wahrscheinlichkeit, dass der Sprinkler die Ursache für das nasse Gras ist. Wissen wir, dass das Gras

WEGERKLÄREN

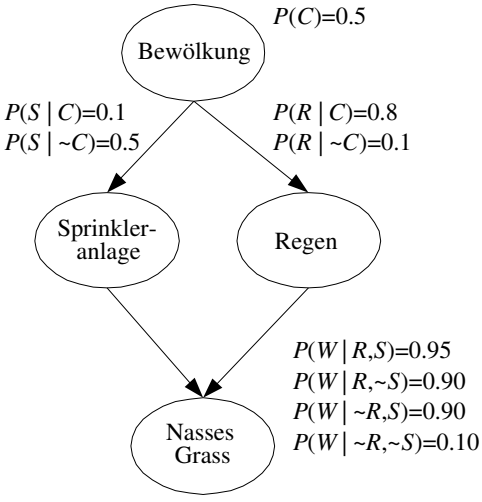


Abb. 14.5: Größere Graphen werden durch die Kombination einfacherer Teilgraphen gebildet, über die unter Verwendung der impliziten bedingten Wahrscheinlichkeiten Information propagiert wird.

nass ist, werden Regen und Sprinkler voneinander abhängig. Entsprechend gilt $P(S|\sim R, W) > P(S|W)$. Dasselbe Verhalten beobachten wir, wenn wir $P(R|W)$ und $P(R|W,S)$ vergleichen (Übung 3).

Wir können größere Graphen konstruieren, indem wir solche Teilgraphen kombinieren. In Abbildung 14.5, in der zwei Teilgraphen kombiniert sind, können wir zum Beispiel die Wahrscheinlichkeit dafür berechnen, dass wir nasses Gras haben, wenn es bewölkt ist:

$$\begin{aligned}
P(W|C) &= \sum_{R,S} P(W,R,S|C) \\
&= P(W,R,S|C) + P(W,\sim R,S|C) \\
&\quad + P(W,R,\sim S|C) + P(W,\sim R,\sim S|C) \\
&= P(W|R,S,C)\,P(R,S|C) \\
&\quad + P(W|\sim R,S,C)\,P(\sim R,S|C) \\
&\quad + P(W|R,\sim S,C)\,P(R,\sim S|C) \\
&\quad + P(W|\sim R,\sim S,C)\,P(\sim R,\sim S|C) \\
&= P(W|R,S)\,P(R|C)\,P(S|C) \\
&\quad + P(W|\sim R,S)\,P(\sim R|C)\,P(S|C) \\
&\quad + P(W|R,\sim S)\,P(R|C)\,P(\sim S|C) \\
&\quad + P(W|\sim R,\sim S)\,P(\sim R|C)\,P(\sim S|C)\,.
\end{aligned}$$

Hierbei haben wir ausgenutzt, dass $P(W|R,S,C) = P(W|R,S)$; wenn R und S gegeben sind, ist W unabhängig von C: Die zwischen ihnen

liegenden Knoten R und S blockieren den Weg zwischen W und C. Ebenso ist $P(R, S|C) = P(R|C)P(S|C)$; wenn C gegeben ist, sind R und S unabhängig. Wir erkennen hier den Vorteil Bayesscher Netze, die Unabhängigkeiten explizit codieren und es gestatten, Inferenzen auf Berechnungen über kleine Gruppen von Variablen zurückzuführen, die von Evidenzknoten zu Abfrageknoten propagiert sind.

Wir berechnen $P(C|W)$ und haben eine diagnostische Inferenz

$$P(C|W) = \frac{P(W|C)\,P(C)}{P(W)} .$$

Die Repräsentation durch einen Graphen ist visuell und ermöglicht ein besseres Verständnis. Das Netz repräsentiert Aussagen über bedingte Wahrscheinlichkeiten und erlaubt es uns, das Problem der Repräsentation der Verbundwahrscheinlichkeit von vielen Variablen auf *lokale* Strukturen zurückzuführen. Das erleichtert sowohl die Analyse als auch die Berechnung. In Abbildung 14.5 ist eine Verbunddichte von vier binären Variablen dargestellt, was normalerweise bedeuten würde, dass fünfzehn $(2^4 - 1)$ Werte gespeichert werden müssen, während wir hier nur neun brauchen. Wenn jeder Knoten nur eine kleine Zahl von Eltern hat, sinkt die Komplexität von exponentiell auf linear (in der Anzahl der Knoten). Wie wir weiter vorn gesehen haben, ist auch das Ziehen von Rückschlüssen einfacher, wenn die Verbunddichte auf bedingte Dichten von kleineren Gruppen von Variablen zurückgeführt wird:

$$P(C, S, R, W) = P(C)\,P(S|C)\,P(R|C)\,P(W|S, R) . \tag{14.11}$$

Im allgemeinen Fall, in dem wir die Variablen $X_1, \ldots, X_d$ haben, schreiben wir

$$P(X_1, \ldots, X_d) = \prod_{i=1}^{d} P(X_i|\text{Elternelemente}(X_i)) . \tag{14.12}$$

Wenn irgendeine Teilmenge von X_i gegeben ist, d. h., wenn ihre Elemente wegen Evidenz auf bestimmte Werte festgelegt sind, dann können wir die Wahrscheinlichkeitsdichte irgendeiner anderen Teilmenge von X_i durch Marginalisieren über die Verbundwahrscheinlichkeit berechnen. Das ist aufwändig, da es die Berechnung einer exponentiellen Zahl von Kombinationen der Verbundwahrscheinlichkeit erfordert, auch wenn jede von ihnen wie in Gleichung 14.11 vereinfacht werden kann. Es ist allerdings zu beachten, dass wir die gleichen Zwischenwerte (Produkte von bedingten Wahrscheinlichkeiten und Summen für das Marginalisieren) verwenden können, wenn die gleiche Evidenz für verschiedene X_i gegeben ist. In Abschnitt 14.5 werden wir den Belief-Propagation-Algorithms kennenlernen, mit dem Interferenz bei geringen Kosten möglich ist, indem die lokalen

Zwischenrechnungen nur einmal ausgeführt werden, die dann mehrere Male für verschiedene Abfrageknoten verwendet werden können.

Zwar verwenden wir in diesem Beispiel binäre Variablen, doch ist es kein großes Problem, die Prozedur auf Fälle zu verallgemeinern, in denen wir diskrete Variablen mit mehreren möglichen Werten haben (bei m möglichen Werten und k Eltern ist für die bedingten Wahrscheinlichkeiten eine Tabelle der Größe m^k nötig). Auch kontinuierliche Variablen sind möglich (in parametrisierter Form, z. B. $p(Y|x) \sim \mathcal{N}(\mu(x|\theta), \sigma^2)$, siehe Abbildung 14.7).

Ein wesentlicher Vorteil von Bayesschen Netzen besteht darin, dass es nicht nötig ist, bestimmte Variablen explizit als Eingabe zu kennzeichnen und andere als Ausgabe. Für jede beliebige Menge von Variablen kann der Wert durch Evidenz festgestellt werden, und die Wahrscheinlichkeiten jeder anderen Menge von Variablen können durch Inferenz ermittelt werden. Der Unterschied zwischen nicht überwachtem und überwachtem Lernen verschwimmt. Aus dieser Perspektive kann ein Graphenmodell als eine „probabilistische Datenbank“ angesehen werden (Jordan 2004), eine Maschine, die Anfragen zu den Werten von Zufallsvariablen beantworten kann.

VERBORGENE VARIABLEN

Bei einem Problem kann es auch *verborgene Variablen* geben, deren Werte niemals durch Evidenz bekannt sind. Die Verwendung von verborgenen Variablen hat den Vorteil, dass die Abhängigkeitsstruktur leichter definiert werden kann. Wenn wir beispielsweise bei der Warenkorbanalyse Abhängigkeiten zwischen den verkauften Artikeln aufdecken wollen, können wir einfach annehmen, dass es eine Abhängigkeit zwischen „Babynahrung“, „Windeln“ und „Milch“ in dem Sinne gibt, dass ein Kunde, der einen dieser Artikel kauft, sehr wahrscheinlich auch die beiden anderen kauft. Anstatt (nichtkausale) Kanten zwischen diese drei Knoten zu setzen, können wir sagen, dass ein verborgener Knoten „Baby im Haushalt“ der verborgene Grund für den Kauf dieser drei Artikel ist. Wenn es verborgene Knoten gibt, werden ihre Werte bei gegebenen Werten der beobachteten Knoten geschätzt und eingetragen.

KAUSALITÄT

Es muss an dieser Stelle darauf hingewiesen werden, dass eine Verbindung von einem Knoten X nicht zwangsläufig eine *Kausalität* bedeutet. Es bedeutet lediglich, dass X direkten Einfluss auf Y hat in dem Sinne, dass die Wahrscheinlichkeit von Y durch den Wert von X bedingt ist, und es kann eine direkte Verbindung zwischen zwei Koten geben, auch wenn es keine direkte Ursache gibt. Es ist vorzuziehen, bei der Konstruktion eines Netzes die kausalen Beziehungen zu haben und eine Erklärung zu liefern, wie die Daten generiert wurden (Pearl 2000), aber solche Ursachen sind nicht immer zugänglich.

14.3 Generative Modelle

Graphenmodelle werden außerdem häufig verwendet, um *generative Modelle* zu visualisieren, die den Prozess repräsentieren, von dem wir glauben, dass er die Daten generiert hat. Für die Klassifikation ist ein Graphenmodell in Abbildung 14.6a gezeigt, wobei $\boldsymbol{x}$ die Eingabe und C eine multinomiale Variable ist, die einen von K Zuständen für den Klassencode annimmt. Es ist, als würden wir zuerst eine Klasse C zufällig auswählen, indem wir eine Stichprobe aus $P(C)$ nehmen, und wenn dann C festgelegt ist, wählen wir ein $\boldsymbol{x}$ aus, indem wir eine Stichprobe aus $p(\boldsymbol{x}|C)$ nehmen. Der Bayessche Satz kehrt die generative Richtung um und erlaubt eine Diagnose, wie in dem Beispiel mit dem Regen und dem nassen Gras, das wir in Abbildung 14.1 betrachtet haben:

GENERATIVES MODELL

$$P(C|\boldsymbol{x}) = \frac{P(C)\,p(\boldsymbol{x}|C)}{P(\boldsymbol{x})} .$$

Im Falle der Clusteranalyse verhält es sich ähnlich, außer dass wir anstelle der multinomialen Variablen C für den Klassenindikator die Variable Z für den Clusterindikator haben und diese nicht während des Trainings beobachtet wird. Der E-Schritt des Expectation-Maximization-Algorithmus (Abschnitt 7.4) verwendet den Bayesschen Satz, um die Kante zu invertieren und trägt bei gegebener Eingabe den Klassenindikator ein.

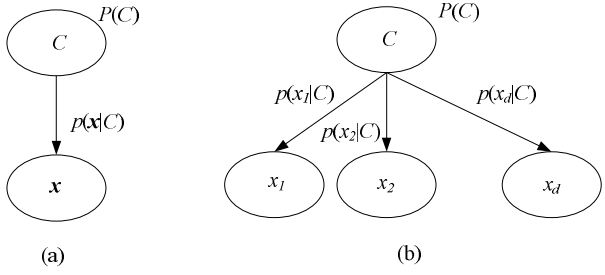


***Abb. 14.6*:** (a) Graphenmodell für die Klassifikation. (b) Naiver Bayesscher Klassifikator, der unabhängige Eingaben annimmt.

Wenn die Eingaben unabhängig sind, erhalten wir den in Abbildung 14.6b gezeigten Graphen, den man den *naiven Bayesschen Klassifikator* nennt, weil er mögliche Abhängigkeiten bzw. Korrelationen zwischen den Eingaben ignoriert, und ein multivariates Problem auf eine Gruppe von univariaten Problemen reduziert:

NAIVER BAYESSCHER KLASSIFIKATOR

$$p(\boldsymbol{x}|C) = \prod_{j=1}^{d} p(x_j|C) .$$

Wir haben die Klassifikation für diesen Fall in den Abschnitten 5.5 (für kontinuierliche $\boldsymbol{x}$) und 5.7 (für diskrete $\boldsymbol{x}$) diskutiert.

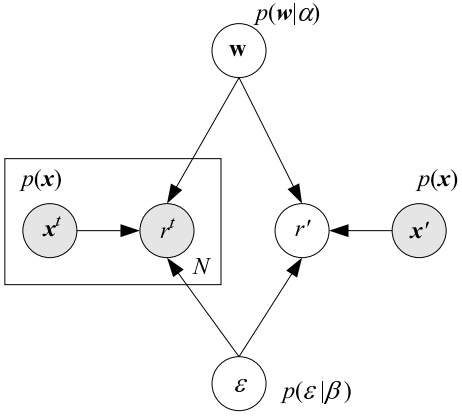


Abb. 14.7: Graphenmodell für die lineare Regression.

Die lineare Regression kann durch ein Graphenmodell visualisiert werden, was in Abbildung 14.7 illustriert ist. Die Eingabe $\boldsymbol{x}^t$ wird aus einer a-priori-Verteilung $p(\boldsymbol{x})$ gezogen, und die abhängige Variable r^t hängt von der Eingabe $\boldsymbol{x}$ und den Gewichten $\boldsymbol{w}$ ab. Hier definieren wir einen Knoten für die Gewichte $\boldsymbol{w}$ mit einer durch α parametrisierten Wahrscheinlichkeit $p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$. Es gibt auch einen Knoten für die Rauschvariable ϵ, parametrisiert durch β, nämlich $p(\epsilon) \sim \mathcal{N}(0, \beta^{-1})$:

$$p(r^t|\boldsymbol{x}^t, \boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}^t, \beta^{-1}) \,. \tag{14.13}$$

Es gibt N solche Paare in der Trainingsmenge, was auf der rechteckigen *Tafel* in der Abbildung dargestellt ist – die Tafel entspricht der Trainingsmenge $\mathcal{X}$. Wenn eine neue Eingabe $\boldsymbol{x}'$ gegeben ist, besteht das Ziel darin, r' zu schätzen. Die Gewichte $\boldsymbol{w}$ sind nicht gegeben, doch sie können mithilfe der Trainingsmenge $\mathcal{X}$ geschätzt werden, die wir in $[\mathbf{X}, \boldsymbol{r}]$ aufteilen.

In Gleichung 14.9, wo C die Ursache für R und S ist, schreiben wir

$$P(R|S) = \sum_C P(R, C|S) = P(R|C)P(C|S) + P(R|\sim C)P(\sim C|S),$$

wobei wir das beobachtete S verwenden und über alle möglichen Werte von C mitteln. Ähnlich schreiben wir hier

$$\begin{aligned} p(r'|\boldsymbol{x}', \boldsymbol{r}, \mathbf{X}) &= \int p(r'|\boldsymbol{x}', \boldsymbol{w})\, p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{r})\, d\boldsymbol{w} \\ &= \int p(r'|\boldsymbol{x}', \boldsymbol{w})\, \frac{p(\mathbf{X}, \boldsymbol{r}|\boldsymbol{w})\, p(\boldsymbol{w})}{p(\mathbf{X}, \boldsymbol{r})}\, d\boldsymbol{w} \\ &\propto \int p(r'|\boldsymbol{x}', \boldsymbol{w}) \prod_t p(r^t|\boldsymbol{x}^t, \boldsymbol{w})\, p(\boldsymbol{w})\, d\boldsymbol{w} \,. \end{aligned} \tag{14.14}$$

Die zweite Zeile folgt durch Anwendung des Satzes von Bayes und die dritte wegen der Unabhängigkeit von Instanzen der Trainingsmenge.

Man beachte, dass es sich bei dem Schema in Abbildung 14.7 um ein Bayessches Modell handelt, in dem wir den Parameter $\boldsymbol{w}$ als eine zufällige Variable mit einer a-priori-Verteilung bezeichnen. Wie wir in Gleichung 14.14 sehen, ist das, was wir effektiv tun, die Schätzung der a-posteriori-Wahrscheinlichkeiten $p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{r})$, über die wir dann intergrieren. Wir haben in Abschnitt 4.4 bereits damit begonnen, dies zu diskutieren und werden uns in Kapitel 16 ausführlicher damit beschäftigen, wobei wir uns verschiedene generative Modelle und unterschiedliche Parametersätze ansehen werden.

14.4 d-Separation

Wir wollen nun das Konzept des Blockierens und Trennens unter der Bezeichnung *d-Separation* verallgemeinern. Dazu definieren wir es in einer Weise, dass wir für beliebige Teilmengen von Knoten A, B und C überprüfen können, ob A und B unabhängig gegeben C sind. Jordan (2004) visualisiert dies durch eine Kugel, die durch den Graphen rollt und bezeichnet diese Visualisierung als *Bayessche Kugel*. Wir setzen die Knoten in C auf ihre Werte, platzieren eine Kugel an jedem Knoten in A, lassen die Kugel nach einem Satz von Regeln durch den Graphen rollen und prüfen, ob eine Kugel irgendeinen Knoten in B erreicht. Wenn dies der Fall ist, sind die Teilmengen abhängig, wenn nicht, sind sie unabhängig.

D-Separation

Bayessche Kugel

Um zu überprüfen, ob A und B d-separiert gegeben C sind, betrachten wir alle möglichen Wege zwischen einem Knoten aus A und einem Knoten aus B. Ein solcher Weg ist blockiert, wenn eine der folgenden Bedingungen erfüllt ist:

(a) Die Richtungen der Kanten entlang des Weges treffen sich entweder Spitze-an-Ende (Fall 1) oder Ende-an-Ende (Fall 2) und der Knoten liegt in C.

(b) Die Richtungen der Kanten entlang des Weges treffen sich Spitze-an-Spitze (Fall 3) und weder dieser Knoten noch irgendeiner seiner Nachkommen liegt in C.

Wenn alle Wege blockiert sind, sagen wir, dass A und B d-separiert sind, d. h. unabhängig gegeben C; andernfalls sind sie abhängig. Beispiele sind in Abbildung 14.8 illustriert.

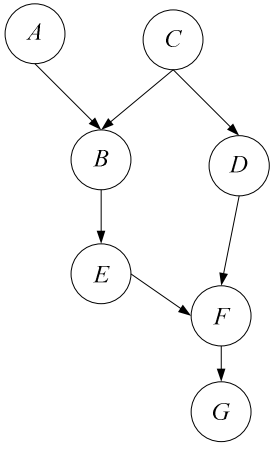


Abb. 14.8: Beispiele für die d-Separation. Der Pfad $BCDF$ ist blockiert durch C, weil C ein Ende-an-Ende-Knoten ist. $BEFG$ ist blockiert durch F, weil F ein Spitze-an-Ende-Knoten ist. $BEFD$ ist blockiert, sofern nicht F (oder G) gegeben ist.

14.5 Belief-Propagation

Nachdem wir uns einige Beispiele für Inferenzen „von Hand“ angesehen haben, sind wir an einem Algorithmus interessiert, der Abfragen wie $P(X|E)$ beantworten kann. Dabei ist X ein beliebiger *Abfrageknoten* des Graphen und E eine beliebige Teilmenge von *Evidenzknoten,* die auf bestimmte Werte festgesetzt sind. Nach Pearl (1988) beginnen wir mit dem einfachsten Fall einer Kette und arbeiten uns allmählich zu komplexeren Graphen vor. Unser Ziel ist es, Graphenoperationen zu finden, die den probabilistischen Prozeduren wie der Bayesschen Marginalisierungsregel entsprechen. Wenn wir das erreichen, können wir die Aufgabe der Inferenz auf allgemeine Graphenalgorithmen abbilden.

14.5.1 Ketten

Eine *Kette* ist eine Folge von Spitze-an-Ende-Knoten, wobei es einen *Wurzelknoten* gibt, der keinen Elternknoten hat. Alle anderen Knoten haben genau einen Elternknoten, und alle Knoten außer dem letzten – dem *Blatt* – haben genau einen Kindknoten. Wenn es in den Vorfahren von X Evidenz gibt, können wir eine diagnostische Inferenz vornehmen und die Evidenz in der Kette abwärts propagieren. Wenn es Evidenz in den Nachkommen von X gibt, können wir eine kausale Inferenz vornehmen und mithilfe des Bayesschen Satzes aufwärts propagieren. Sehen wir uns nun den allgemeinen Fall an, in dem es Evidenz in beiden Richtungen gibt, die Kette aufwärts E^+ und die Kette abwärts E^- (siehe Abbildung 14.9). Dabei ist zu beachten, dass jeder Evidenzknoten den Knoten X von allen Knoten auf der anderen Seite der Evidenz separiert und dass deren Werte keinen Einfluss auf $p(X)$ haben. Dies gilt in beiden Richtungen.

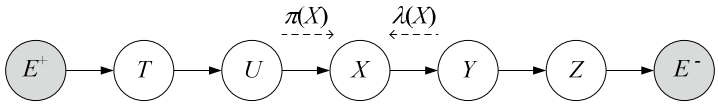


***Abb. 14.9*:** Inferenz entlang einer Kette.

Wir fassen jeden Knoten als einen Prozessor auf, der Nachrichten von seinen Nachbarn empfängt und diese nach gewissen lokalen Berechnungen weitergibt. Jeder Knoten X berechnet und speichert lokal zwei Werte: $\lambda(X) \equiv P(E^-|X)$ ist das propagierte E^-, das X von seinem Kind empfängt und an seinen Elter weitergibt; $\pi(X) \equiv P(X|E^+)$ ist das propagierte E^+, das X von seinem Elter empfängt und an sein Kind weitergibt. Damit haben wir

$$\begin{aligned} P(X|E) &= \frac{P(E|X)\,P(X)}{P(E)} = \frac{P(E^+,E^-|X)\,P(X)}{P(E)} \\ &= \frac{P(E^+|X)\,P(E^-|X)\,P(X)}{P(E)} \\ &= \frac{P(X|E^+)\,P(E^+)\,P(E^-|X)\,P(X)}{P(X)\,P(E)} \\ &= \alpha P(X|E^+)\,P(E^-|X) = \alpha\pi(X)\lambda(X)\,, \end{aligned} \tag{14.15}$$

wobei α eine Normierungskonstante ist, die nicht von dem Wert von X abhängt. Die zweite Zeile ergibt sich, weil E^+ und E^- unabhängig gegeben X sind, und für die dritte Zeile wurde der Bayessche Satz ausgenutzt.

Wenn für einen Knoten E ein bestimmter Wert $\bar{e}$ realisiert wird, ist $\lambda(\bar{e}) \equiv 1$ und $\lambda(e) \equiv 0$ für $e \neq \bar{e}$. Für den Blattknoten X, der nicht instanziiert ist, gilt für alle x-Werte $\lambda(x) \equiv 1$. Der Wurzelknoten, der ebenfalls nicht instanziiert ist, nimmt die a-priori-Wahrscheinlichkeiten als π-Werte an: $\pi \equiv P(x), \forall x$.

Mit diesen Anfangsbedingungen können wir Rekursionsformeln ableiten, um die Evidenz entlang der Kette zu propagieren.

Für π-Nachrichten haben wir

$$\begin{aligned} \pi(X) \equiv P(X|E^+) &= \sum_U P(X|U,E^+)\,P(U|E^+) \\ &= \sum_U P(X|U)\,P(U|E^+) = \sum_U P(X|U)\,\pi(U)\,, \end{aligned} \tag{14.16}$$

wobei die zweite Zeile aus der Tatsache folgt, dass U den Weg zwischen X und E^+ blockiert.

Für die λ-Nachrichten haben wir

$$\begin{aligned}\lambda(X) &\equiv P(E^-|X) = \sum_Y P(E^-|X,Y)\,P(Y|X) \\ &= \sum_Y P(E^-|Y)\,P(Y|X) = \sum_Y P(Y|X)\,\lambda(Y)\,, \end{aligned} \tag{14.17}$$

wobei die zweite Zeile aus der Tatsache folgt, dass Y den Weg zwischen X und E^- blockiert.

Wenn die Evidenzknoten auf einen Wert festgelegt werden, denn initiieren sie Traffic und die Knoten werden solange aktualisiert, bis es Konvergenz gibt. Pearl (1988) betrachtet dies als eine parallele Maschine, in der jeder Knoten durch einen Prozessor implementiert wird, der parallel mit anderen arbeitet und über λ- und π-Nachrichten mit seinem Elter und seinem Kind Informationen austauscht.

14.5.2 Bäume

Die Kette ist ein beschränktes Modell, da jeder Knoten nur einen einzigen Elter und ein einziges Kind haben kann, also eine einzige Ursache und ein einziges Symptom. In einem *Baum* dagegen kann jeder Knoten mehrere Kinder haben, aber außer dem Wurzelknoten hat jeder weiterhin genau einen einzigen Elter. Wir können die gleiche Belief-Propagation wie zuvor auch hier anwenden, wobei im Unterschied zur Kette jeder Knoten verschiedene λ-Nachrichten von seinen Kindern empfängt und verschiedene π-Nachrichten an seine Kinder versendet. Wir bezeichen also mit $\lambda_Y(X)$ die Nachricht, die X von Kind Y erhält und mit $\pi_Y(X)$ die Nachricht, die X an Kind Y sendet.

Wieder teilen wir die mögliche Evidenz in zwei Teile auf: E^- sind die Knoten, deren Wurzel im Teilbaum der Abfrageknoten X ist, und E^+ sind die übrigen Evidenzknoten (siehe Abbildung 14.10). Man beachte, dass die Knoten dieser zweiten Gruppe keine Vorfahren von X sein müssen, sondern auch zu einem Teilbaum gehören können, dessen Wurzel ein Geschwister von X ist. Der entscheidende Punkt ist, dass X wie zuvor E^+ und E^- separiert, so dass wir schreiben können $P(E^+, E^-|X) = P(E^+|X)\,P(E^-|X)$. Damit haben wir

$$P(X|E) = \alpha\pi(X)\lambda(X)\,,$$

wobei α wieder eine Normierungskonstante ist.

$\lambda(X)$ ist die Evidenz in dem Teilbaum, dessen Wurzel X ist, und wenn X wie in Abbildung 14.10 zwei Kinder Y und Z hat, kann diese wie folgt berechnet werden:

$$\begin{aligned}\lambda(X) &\equiv P(E_X^-|X) = P(E_Y^-, E_Z^-|X) \\ &= P(E_Y^-|X)\,P(E_Z^-|X) = \lambda_Y(X)\lambda_Z(X)\,. \end{aligned} \tag{14.18}$$

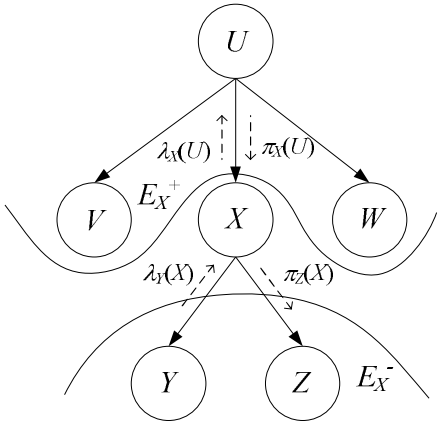


***Abb. 14.10*:** In einem Baum kann jeder Knoten mehrere Kinder haben, jedoch nur einen Elter.

Im allgemeinen Fall kann X m Kinder Y_j, $j = 1, \ldots, m$, haben und wir müssen dann alle λ-Werte multiplizieren:

$$\lambda(X) = \prod_{j=1}^{m} \lambda_{Y_j}(X) \,. \tag{14.19}$$

Nachdem X die Evidenz λ von den λ-Nachrichten seiner Kinder akkumuliert hat, propagiert diese aufwärts zu seinem Elter:

$$\lambda_X(U) = \sum_X \lambda(X)\, P(X|U) \,. \tag{14.20}$$

Entsprechend ist in der anderen Richtung $\pi(X)$ die übrige Evidenz, die in $P(U|E^+)$ akkumuliert ist und die als π-Nachricht an X weitergegeben wird:

$$\pi(X) \equiv P(X|E_X^+) = \sum_U P(X|U)\, P(U|E_X^+) = \sum_U P(X|U)\, \pi_X(U). \tag{14.21}$$

Dieser berechnete π-Wert wird dann abwärts zu den Kindern von X propagiert. Man beachte, dass das, was Y von X empfängt, das ist, was X von seinem Elter U sowie von seinem anderen Kind Z empfängt: zusammen macht das E_Y^+ (siehe Abbildung 14.10):

$$\begin{aligned} \pi_Y(X) &\equiv P(X|E_Y^+) = P(X|E_X^+, E_Z^-) \\ &= \frac{P(E_Z^-|X, E_X^+)\, P(X|E_X^+)}{P(E_Z^-)} = \frac{P(E_Z^-|X)\, P(X|E_X^+)}{P(E_Z^-)} \\ &= \alpha \lambda_Z(X) \pi(X) \,. \end{aligned} \tag{14.22}$$

Wenn Y nicht nur ein Geschwister Z hat, sondern viele, gilt auch hier wieder, dass wir ein Produkt über alle ihre λ-Werte bilden müssen:

$$\pi_{Y_j}(X) = \alpha \prod_{s \neq j} \lambda_{Y_S}(X)\pi(X) \,. \tag{14.23}$$

14.5.3 Mehrfachbäume

Mehrfachbaum

In einem Baum hat jeder Knoten einen einzigen Elter, d. h. eine einzige Ursache. In einem *Mehrfachbaum* kann ein Knoten dagegen mehrere Eltern haben, wobei wir aber fordern, dass der Graph einfach zusammenhängend ist, d. h., es darf zwischen jedem Paar von Knoten nur eine Kette geben. Wenn wir X entfernen, wird der Graph in zwei Komponenten geteilt. Das ist notwendig, damit wir damit fortfahren können, E_X in E_X^+ und E_X^- zu teilen, die unabhängig gegeben X sind (siehe Abbildung 14.11).

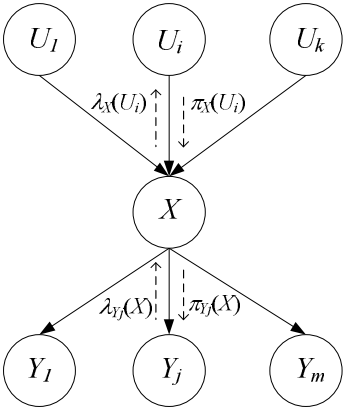


Abb. 14.11: In einem Mehrfachbaum kann ein Knoten mehrere Kinder und mehrere Eltern haben. Dabei ist der Graph noch immer einfach verbunden, d. h., es gibt eine einzige Kette zwischen U_i und Y_j, die durch X geht.

Wenn der Knoten X mehrere Eltern U_i, $i = 1, \ldots, k$ hat, empfängt er π-Nachrichten von ihnen allen, und diese $\pi_X(U_i)$ kombiniert er wie folgt:

$$\begin{aligned}
\pi(X) \equiv P(X|E_X^+) &= P(X, E_{U_1X}^+, E_{U_2X}^+, \ldots, E_{U_kX}^+) \\
&= \sum_{U_1} \sum_{U_2} \cdots \sum_{U_k} P(X|U_1, U_2, \ldots, U_k)\, P(U_1|E_{U_1X}^+) \ldots P(U_k|E_{U_kX}^+) \\
&= \sum_{U_1} \sum_{U_2} \cdots \sum_{U_k} P(X|U_1, U_2, \ldots, U_k) \prod_{i=1}^{k} \pi_X(U_i) \,.
\end{aligned} \tag{14.24}$$

Das Ergebnis sendet er weiter an die verschiedenen Kinder Y_j, $j = 1, \ldots, m$:

$$\pi_{Y_j}(X) = \alpha \prod_{s \neq j} \lambda_{Y_s}(X)\pi(X) \,. \tag{14.25}$$

Wenn X mehrere Eltern hat, kombiniert sich eine λ-Nachricht, die X an einen Elter U_i sendet, nicht nur mit der Evidenz, die X von seinen Kindern empfängt, sondern auch mit den π-Nachrichten, die X von seinen anderen Eltern $U_r, r \neq i$ empfängt. Zusammen ergeben sie $E_{\bar{U}_i,X}$:

$$\begin{aligned}
\lambda_X(U_i) &\equiv P(E^-_{U_i X}|X) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X, E^+_{U_{r\neq i}X}, X, U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X, E^+_{U_{r\neq i}X}|X, U_{r\neq i}, U_i)\, P(X, U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X|X) P(E^+_{U_{r\neq i}X}|U_{r\neq i}) P(X|U_{r\neq i}) P(U_{r\neq i}|U_i) \\
&= \sum_X \sum_{U_{r\neq i}} P(E^-_X|X) \frac{P(U_{r\neq i}|E^+_{U_{r\neq i}X})\, P(E^+_{U_{r\neq i}X})}{P(U_{r\neq i})} \\
&\qquad\qquad \times P(X|U_{r\neq i}, U_i)\, P(U_{r\neq i}|U_i) \\
&= \beta \sum_X \sum_{U_{r\neq i}} P(E^-_X|X)\, P(U_{r\neq i}|E^+_{U_{r\neq i}X})\, P(X|U_{r\neq i}, U_i) \\
&= \beta \sum_X \sum_{U_{r\neq i}} \lambda(X) \prod_{r\neq i} \pi_X(U_r)\, P(X|U_1, \ldots, U_k) \\
&= \beta \sum_X \lambda(X) \sum_{U_{r\neq i}} P(X|U_1, \ldots, U_k) \prod_{r\neq i} \pi_X(U_r)\,. \qquad (14.26)
\end{aligned}$$

Um den Gesamtwert von λ zu finden, multipliziert der Elter wie bei einem Baum die λ-Nachrichten, die er von seinen Kindern empfängt:

$$\lambda(X) = \prod_{j=1}^{m} \lambda_{Y_j}(X)\,. \qquad (14.27)$$

In dem hier betrachteten Fall, in dem es mehrere Eltern geben kann, müssen wir die bedingten Wahrscheinlichkeiten gegeben alle Eltern, $p(X|U_1, \ldots, U_k)$, speichern und verarbeiten, was für große k hohe Kosten bedeutet. Es existieren Vorschläge, wie sich die Komplexität von exponentiell in k auf linear reduzieren lässt. In einem verrauschten OR-Gatter zum Beispiel ist jeder Elter in der Lage, das Ereignis zu verursachen und die Likelihood verringert sich nicht, wenn Mehrelter-Ereignisse auftreten. Wenn die Wahrscheinlichkeit, dass X auftritt, wenn nur die Ursache U_i auftritt, $1 - q_i$ ist, also

$$P(X|U_i, {\sim}U_{p\neq j}) = 1 - q_i\,, \qquad (14.28)$$

dann berechnet sich die Wahrscheinlichkeit, dass X auftritt, wenn eine Teilmenge T von ihnen vorkommt, durch

$$P(X|T) = 1 - \prod_{u_i \in T} q_i \,. \tag{14.29}$$

Nehmen wir zum Beispiel an, dass nasses Gras zwei mögliche Gründe hat – Regen oder die Sprinkleranlage – wobei $q_R = q_S = 1$ ist. Das heißt, jeder der beiden Gründe hat für sich allein eine 90-prozentige Wahrscheinlichkeit, nasses Gras zu verursachen. Damit ist $P(W|R, \sim S) = 0{,}9$ und $P(W|R, S) = 0{,}99$.

Eine weitere Möglichkeit besteht darin, die bedingte Wahrscheinlichkeit durch eine Funktion auszudrücken, wobei ein Satz von Parametern gegeben ist. Beispielsweise kann sie als lineares Modell

$$P(X|U_1, \ldots, U_k, w_0, w_1, \ldots, w_k) = \text{sigmoid}\left(\sum_{k=1}^{k} w_i U_i + w_0\right) \tag{14.30}$$

geschrieben werden, wobei die Sigmoidfunktion sicherstellt, dass die Ausgabe eine Wahrscheinlichkeit, d. h. eine Zahl zwischen 0 und 1 ist. Während des Trainings können wir die Parameter $w_i, i = 0, \ldots, d$ lernen, etwa um die Likelihood über eine Stichprobe zu maximieren.

14.5.4 Verbindungsbäume

Wenn es in dem betrachteten ungerichteten Graphen eine Schleife gibt – beispielsweise wenn die Eltern von X einen gemeinsamen Vorfahren haben –, funktioniert der zuvor diskutiert Algorithmus nicht mehr. In einem solchen Fall gibt es mehr als einen Weg, auf dem Evidenz propagieren kann und wir können, etwa bei der Auswertung der Wahrscheinlichkeit bei X, nicht sagen, dass X die Evidenz E in einen kausalen (aufwärts gerichteten) Teil E_X^+ und einen diagnostischen (abwärts gerichteten) Teil E_X^- separiert; das Entfernen von X führt nicht dazu, dass der Graph in zwei Teile zerfällt. Die Bedingtheit von X macht die Teile nicht unabhängig und sie können über irgendeinen anderen Weg, der X nicht beinhaltet, interagieren.

Wir können weiterhin den gleichen Algorithmus anwenden, wenn es möglich ist, den Graphen in einen Mehrfachbaum zu konvertieren. Wir definieren Cliquenknoten, die einer Menge von Originalvariablen entsprechen und diese so verbinden, dass sie einen Baum bilden (siehe Abbildung 14.12). Dann können wir den oben vorgestellten Algorithmus für die Belief-Propagation anwenden, wenn wir ihn etwas modifizieren. Das ist im Wesentlichen die Idee hinter dem *Verbindungsbaum-Algorithmus* (Lauritzen und Spiegelhalter 1988; Jensen 1996; Jordan 2004).

VERBINDUNGSBAUM

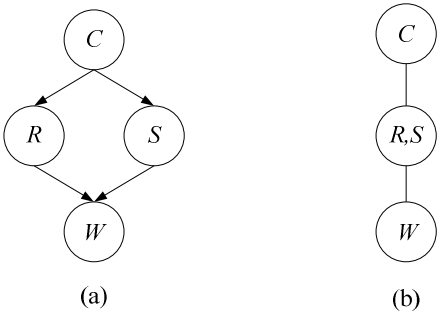


Abb. 14.12: (a) Ein mehrfach zusammenhängender Graph und (b) der dazugehörige Verbindungsbaum mit geclusterten Knoten.

14.6 Ungerichtete Graphen: Markovsche Zufallsfelder

Bis jetzt haben wir uns mit gerichteten Graphen befasst, bei denen die Einflüsse ungerichtet sind, und haben dabei den Satz von Bayes angewendet, um die Kanten zu invertieren. Wenn die Einflüsse symmetrisch sind, repräsentieren wir sie mithilfe eines ungerichteten Graphenmodells, das auch als *Markovsches Zufallsfeld* bekannt ist. Beispielsweise haben benachbarte Pixel in einem Bild die Tendenz, die gleiche Farbe zu haben – sie sind also korreliert –, und diese Korrelation geht in beide Richtungen.

MARKOVSCHES ZUFALLSFELD

Durch gerichtete Graphen wird der Begriff der bedingten Unabhängigkeit anders definiert als durch ungerichtete Graphen. Aus diesem Grund gibt es Wahrscheinlichkeitsdichten, die durch einen gerichteten Graphen und nicht durch einen ungerichteten repräsentiert werden – und umgekehrt (Pearl 1988).

Da die Kanten bei ungerichteten Graphen keine Richtung haben und somit nicht zwischen Spitze und Ende unterschieden werden kann, ist die Behandlung von ungerichteten Graphen einfacher. Beispielsweise ist es viel einfacher zu prüfen, ob A und B unabhängig gegeben C sind. Wir prüfen einfach, ob es nach dem Entfernen aller Knoten von C noch einen Weg zwischen einem Knoten aus A und einem Knoten aus B gibt. Ist dies der Fall, dann sind sie abhängig. Andernfalls laufen alle Wege zwischen Knoten von A und Knoten von B über Knoten von C, so dass durch das Entfernen von C zwei separate Komponenten für die Knoten von A und die Knoten von B entstehen. Dann haben wir Unabhängigkeit.

Bei ungerichteten Graphen sprechen wir nicht von Eltern oder Kindern, sondern von *Cliquen*. Darunter verstehen wir Knotenmengen, die die Eigenschaften haben, dass es zwischen jedem Paar von Knoten eine Verbindung gibt. Eine *maximale* Clique hat die größtmögliche Anzahl von Elementen. Anstelle von bedingten Wahrscheinlichkeiten (die eine Richtung implizieren) haben wir in ungerichteten Graphen *Potenzialfunktionen* $\psi_C(X_C)$, wobei X_C die Menge der Variablen in der Clique

CLIQUE

POTENZIALFUNKTION

C ist, und wir definieren die Verbundverteilung als das Produkt der Potenzialfunktionen der maximalen Cliquen des Graphen:

$$p(X) = \frac{1}{Z} \prod_C \Psi_C(X_C) \,. \tag{14.31}$$

Dabei ist Z die Normierungskonstante, die $\sum_X p(X) = 1$ sicherstellt:

$$Z = \sum_X \prod_C \Psi_C(X) \,. \tag{14.32}$$

Man kann zeigen, dass ein gerichteter Graph bereits normiert ist (Übung 5).

Anders als bei gerichteten Graphen muss es für die Potenzialfunktionen von ungerichteten Graphen keine probabilistische Interpretation geben, und man hat größere Freiheiten bei ihrer Definition. Allgemein können wir Potenzialfunktionen als Ausdruck von lokalen Nebenbedingungen ansehen, d. h., sie favorisieren bestimmte lokale Konfigurationen vor anderen. Beispielsweise können wir für ein Bild eine paarweise Potenzialfunktion zwischen benachbarten Punkten definieren, die für ähnliche Farben der Pixel einen höheren Wert annimmt als für verschiedene Farben (Bishop 2006). Wenn wir dann für einige Pixel die Werte als Evidenz vorgeben, können wir für andere Pixel die Farbwerte schätzen, beispielsweise für solche, die verdeckt und daher unbekannt sind.

Wenn wir den gerichteten Graphen haben, ist es einfach, ihn als ungerichteten Graphen neu zu zeichnen. Dazu müssen wir nur alle Richtungen weglassen, und wenn ein Knoten einen einzigen Elter hat, können wir die paarweise Potenzialfunktion einfach gleich der bedingten Wahrscheinlichkeit setzen. Wenn der Knoten mehr als einen Elter hat, bewirkt allerdings das Phänomen des „Wegerklärens“ für Spitze-an-Spitze-Knoten, dass die Eltern abhängig sind. Folglich sollten die Eltern zur selben Clique gehören, so dass das Cliquenpotenzial alle Eltern umfasst. Das geschieht, indem alle Eltern eines Knotens durch eine Kante verbunden werden, so dass sie vollständig untereinander verbunden sind und eine Clique bilden. Man sagt, die Eltern werden „verheiratet“, und der gesamte Prozess wird *Moralisieren* genannt. Das Moralisieren ist übrigens ein Schritt beim Generieren eines Verbindungsbaums, bei dem es sich um einen ungerichteten Graphen handelt.

Moralisieren

Es ist nicht schwierig, den Algorithmus der Belief-Propagation so anzupassen, dass er auf ungerichteten Graphen verwendet werden kann. Er wird dann sogar einfacher, weil die Potenzialfunktion symmetrisch ist und wir nicht zwischen kausaler und diagnostischer Evidenz unterscheiden müssen. Auf ungerichteten Ketten und Bäumen ist somit Interferenz möglich. In Mehrfachbäumen dagegen, wo ein Knoten mehrere Eltern hat und Moralisieren zwangsläufig Schleifen erzeugt, würde das nicht

funktionieren. Hier kann man den Trick anwenden, den Graphen in einen *Faktorgraphen* zu konvertieren, der zusätzlich zu den Variablenknoten eine zweite Art von Knoten verwendet, die *Faktorknoten* genannt werden. Dann schreiben wir die Verbundverteilung als ein Produkt von Faktoren (Kschischang, Frey und Loeliger 2001):

FAKTORGRAPH

$$p(X) = \frac{1}{Z} \prod_S f_S(X_S) \,. \tag{14.33}$$

Dabei bezeichnet X_S eine Teilmenge der Variablenknoten, die von dem Faktor S verwendet wird. Gerichtete Graphen sind ein Spezialfall, bei dem die Faktoren lokalen bedingten Wahrscheinlichkeiten entsprechen, Ungerichtete Graphen sind ein anderer Spezialfall, bei dem die Faktoren Potenzialfunktionen über maximale Cliquen sind. Der Vorteil besteht darin, dass, wie man in Abbildung 14.13 sieht, die Baumstruktur auch nach der Moralisierung erhalten bleibt.

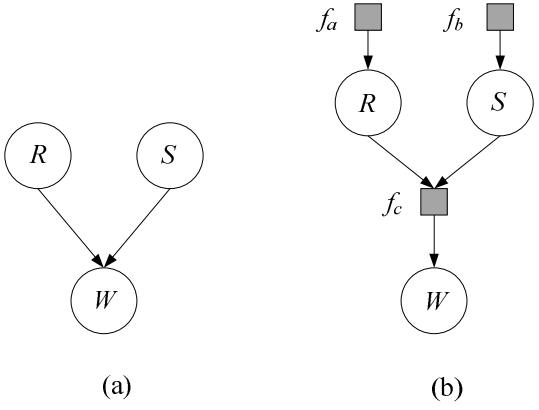


***Abb. 14.13*:** (a) Ein gerichteter Graph, der durch Moralisieren eine Schleife bekommen würde; (b) der zugehörige Faktorgraph, der ein Baum ist. Die drei Faktoren sind $f_a(R) \equiv P(R)$, $f_b(R) \equiv P(S)$ und $f_a(R, S, W) \equiv P(W|R, S)$.

Der Belief-Propagation-Algorithmus kann so verallgemeinert werden, dass er auch auf Faktorgraphen anwendbar ist. Diese Verallgemeinerung wird *Summen-Produkt-Algorithmus* genannt (Bishop 2006; Jordan 2004), und die Idee ist wieder, dass lokale Berechnungen einmal ausgeführt und diese dann als Nachrichten durch den Graphen propagiert werden. Der Unterschied hier ist, dass es wegen der beiden Arten von Knoten auch zwei Arten von Nachrichten gibt, und wir unterscheiden zwischen diesen beiden Arten. Dabei ist zu beachten, dass ein Faktorgraph bipartit ist und ein Knoten der einen Art nur mit Knoten der anderen Art eine enge Verbindung haben kann.

SUMMEN-PRODUKT-ALGORITHMUS

Das Ziel besteht bei der Belief-Propagation bzw. beim Summen-Faktor-Algorithmus darin, die Wahrscheinlichkeit einer Menge von Knoten X zu

finden, wenn eine andere Menge von Evidenzknoten E auf einen bestimmten Wert festgesetzt wurde, also $P(X|E)$. Bei manchen Anwendungen sind wir daran interessiert, die Einstellung aller X zu finden, welche die vollständige Verbundwahrscheinlichkeitsverteilung $p(X)$ maximiert. Im ungerichteten Fall zum Beispiel, wo Potenzialfunktionen lokal konsistente Konfigurationen codieren, würde ein solcher Ansatz lokale Nebenbedingungen über den gesamten Graphen propagieren und eine Lösung finden, die die globale Konsistenz maximiert. In einem Graphen, dessen Knoten Pixeln entsprechen und paarweise Potenzialfunktionen Korrelationen favorisieren, kann mit diesem Ansatz eine Rauschunterdrückung implementiert werden (Bishop 2006). Der Algorithmus hierfür wird *max-Produkt-Algorithmus* genannt (Bishop 2006; Jordan 2004) und er ist im Wesentlichen der gleiche wie der Summen-Produkt-Algorithmus, mit dem Unterschied, dass in diesem Fall das Maximum (der wahrscheinlichste Wert) genommen wird anstatt die Summe zu bilden (Marginalisieren). Das ist analog zu dem Unterschied zwischen der Vorwärts-Rückwärts-Prozedur und dem Viterbi-Algorithmus in Hidden-Markov-Modellen, die wir in Kapitel 15 diskutieren werden.

Max-Produkt-Algorithmus

Es sei angemerkt, dass Knoten nicht mit Konzepten auf niedriger Beschreibungsebene wie etwa Pixeln korrespondieren müssen. Beispielsweise kann es bei einer Anwendung auf dem Gebiet des maschinellen Sehens Knoten für Ecken unterschiedlicher Art oder für Linien unterschiedlicher Richtungen geben, wobei Potenzialfunktionen zum Überprüfen der Kompatibilität verwendet werden, um etwa festzustellen, ob sie Teil der gleichen Interpretation sein können – man denke zum Beispiel an den Necker-Würfel – so dass man nach der Konsolidierung lokaler Evidenzen global konsistente Lösungen hat.

Die Komplexität von Inferenz-Algorithmen auf Mehrfachbäumen oder Verbindungsbäumen wird durch die maximale Zahl von Eltern bzw. den Umfang der größten Clique bestimmt, und wenn dieser Wert groß ist, kann es sein, dass exakte Inferenz nicht mehr möglich ist. In diesem Fall wird man eine Approximation oder einen Sampling-Algorithmus einsetzen müssen (Jordan 1999; Bishop 2006).

14.7 Lernen der Struktur eines Graphenmodells

Wie jedes andere Verfahren besteht auch das Lernen eines Graphenmodells aus zwei Teilen. Der erste Teil ist das Lernen von Parametern bei gegebener Struktur, was relativ einfach ist (Buntine 1996). In Graphenmodellen können Tabellen für bedingte Wahrscheinlichkeiten oder ihre Parametrisierungen (wie in Gleichung 14.30) darauf trainiert werden, die Likelihood zu maximieren, oder es wird ein Bayesscher Ansatz verwendet, falls die a-priori-Wahrscheinlichleiten bekannt sind (Kapitel 16).

Im zweiten Teil, der schwieriger, aber interessant ist, wird die Struktur des Graphen gelernt (Cowell et al. 1999). Dies ist im Wesentlichen ein Modellauswahlproblem, und ganz ähnlich wie bei den inkrementellen Ansätzen für das Lernen der Struktur von mehrlagigen Perzeptronen (Abschnitt 11.9) können wir dieses als eine Suche im Raum aller möglichen Graphen auffassen. Man kann zum Beispiel Operationen betrachten, die Kanten und/oder verborgene Knoten hinzufügen oder entfernen und dann eine Suche durchführen, bei der die Verbesserung in jedem Schritt ausgewertet wird (dabei werden in jeder Iteration Parameter gelernt). Es ist allerdings zu beachten, dass man zum Prüfen auf Überanpassung richtig regularisieren muss, was einem Bayesschen Ansatz mit einem a-priori-Wert entspricht, der einfachere Graphen favorisiert (Neapolitan 2004). Da aber der Zustandsraum groß ist, kann hier am ehesten ein menschlicher Experte Abhilfe schaffen, der manuell Kausalbeziehungen zwischen den Variablen definiert und Teilgraphen von kleinen Gruppen von Variablen erzeugt.

In Kapitel 16 diskutieren wir den Bayesschen Ansatz und in Abschnitt 16.8 speziell die nichtparametrischen Bayesschen Verfahren, bei denen die Modellstruktur zeitlich komplexer gemacht werden kann, wenn mehr Daten eintreffen.

14.8 Einflussdiagramme

Ähnlich wie in Kapitel 3, wo wir von Wahrscheinlichkeiten auf Aktionen mit Risiken verallgemeinert haben, können wir Graphenmodelle zu *Einflussdiagrammen* erweitern, die es uns gestatten, Entscheidungen und Hilfsgrößen mit einzubeziehen. Ein Einflussdiagramm umfasst *Zufallsknoten*, welche die in Graphenmodellen verwendeten Zufallsvariablen repräsentieren (siehe Abbildung 14.14). Es besitzt außerdem Entscheidungsknoten und einen Hilfsknoten. Ein Entscheidungsknoten repräsentiert eine Auswahl von Aktionen. Einen Hilfsknoten gibt es dort, wo die Hilfsgröße berechnet wird. Entscheidungen können auf Zufallsknoten basieren und sie können andere Zufallsknoten sowie den Hilfsknoten beeinflussen.

EINFLUSSDIAGRAMME

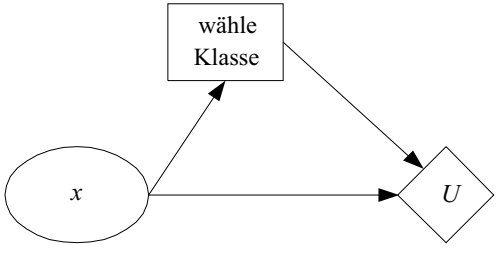


Abb. 14.14: Einflussdiagramm, das der Klassifikation entspricht. In Abhängigkeit von der Eingabe x wird eine Klasse gewählt, die eine bestimmte Hilfsgröße (Unsicherheit) hat.

Inferenz auf einem Einflussdiagramm ist eine Erweiterung der Belief-Propagation auf einem Graphenmodell. Wenn Evidenz auf einigen der Zufallsknoten gegeben ist, wird diese Evidenz propagiert; für jede mögliche Entscheidung wird die Hilfsfunktion berechnet und dann die Entscheidung mit dem größten Hilfswert ausgewählt. Das Einflussdiagramm für die Klassifikation einer gegebenen Eingabe ist in Abbildung 14.14 gezeigt. Bei gegebener Eingabe entscheidet der Entscheidungsknoten über die Klasse, und für jede Entscheidung haben wir eine bestimmte Hilfsgröße (Unsicherheit).

14.9 Anmerkungen

Graphenmodelle haben zwei Vorteile. Der erste besteht darin, dass wir die Beziehungen zwischen den Variablen visualisieren können und so ein besseres Verständnis von dem betrachteten Problem gewinnen, beispielsweise indem wir ein kausales generatives Modell verwenden. Der zweite Vorteil ist der, dass wir durch das Bestimmen von Graphenoperationen, die den grundlegenden probabilistischen Prozeduren wie dem Satz von Bayes oder dem Marginalisieren entsprechen, das Problem der Inferenz auf allgemeine Graphenalgorithmen abbilden können, die sehr effizient repräsentiert und implementiert werden können.

Die Idee der visuellen Repräsentation von Variablen und den zwischen ihnen bestehenden Abhängigkeiten durch einen Graphen und die damit verwandte Idee der Faktorisierung einer komplizierten globalen Funktion von vielen Variablen in ein Produkt aus lokalen Funktionen, die jeweils nur von einer kleinen Teilmenge dieser Variablen abhängen, wird offenbar in unterschiedlichen Bereichen der Entscheidungsfindung, des Codierens und der Signalverarbeitung angewendet. Eine Übersicht ist in Kschischang, Frey und Loeliger (2001) zu finden.

Die Komplexität der Inferenzalgorithmen auf Mehrfachbäumen oder Verbindungsbäumen wird durch die maximale Anzahl von Eltern bzw. vom Umfang der größten Clique bestimmt. Wenn diese Anzahl groß ist, kann es passieren, dass exakte Inferenz nicht mehr möglich ist. In einem solchen Fall muss man auf eine Approximation oder einen Sampling-Algorithmus zurückgreifen. Variationsverfahren und Markov-Ketten-Monte-Carlo-Algorithmen (MCMC) werden in Jordan et al. 1999, MacKay 2003, Andrieu et al. 2003, Bishop 2006 sowie Murphy 2012 diskutiert.

Graphenmodelle sind besonders geeignet, um Bayessche Ansätze zu repräsentieren, wobei wir zusätzlich zu den Knoten für beobachtete Variablen noch Knoten für verborgene Variablen und die Modellparameter haben. Wir können auch eine Hierarchie einführen, in der wir Knoten für Hyperparameter haben – also Parameter auf der zweiten Beschreibungsebene für die a-priori-Parameter der ersten Beschreibungsebene.

Es kann das Verständnis und auch die Inferenz in vielen Anwendungsgebieten erleichtern, wenn man sich die Daten als Stichproben aus einem kausalen generativen Modell vorstellt, das als Graph visualisiert werden kann. Bei der Kategorisierung von Texten kann man sich zum Beispiel einen Text als einen Prozess vorstellen, bei dem sich ein Autor zunächst entscheidet, ein Dokument über eine Reihe von Themen zu schreiben und dann für jedes Thema eine Menge von Wörtern auswählt. In der Bioinformatik werden Graphen unter anderem bei der Modellierung von *phylogenetischen Bäumen* benutzt. Dabei handelt es sich um gerichtete Graphen, in denen lebende Arten durch Blätter repräsentiert werden, während Knoten, die keine Endknoten sind, Vorfahren repräsentieren, die sich im Zuge der Artenbildung irgendwann in unterschiedliche Arten aufgespalten haben und deren bedingten Wahrscheinlichkeiten vom evolutionären Abstand zwischen einer Art und ihrem Vorfahren abhängt (Jordan 2004).

PHYLOGENETISCHER BAUM

Das Hidden-Markov-Modell, das wir in Kapitel 15 behandeln werden, ist ebenfalls ein Graphenmodell. Bei diesem Modell sind die Eingaben sequenziell abhängig, wie etwa bei der Spracherkennung, wo ein Wort eine spezielle Sequenz von Lauten ist. Diese Sequenzen nennt man Phoneme (Ghahramani 2001). Anwendungen für solche dynamischen Graphenmodelle gibt es in vielen Gebieten, in denen die Zeit als eine Dimension auftritt, so etwa beim Sprechen, in der Musik usw. (Zweig 2003; Bilmes und Bartels 2005).

Graphenmodelle werden auch beim maschinellen Sehen verwendet, beispielsweise beim Information Retrieval (Barnard et al. 2003) und bei der Szenenanalyse (Sudderth et al. 2008). Eine Übersicht über die Verwendung von Graphenmodellen in der Bioinformatik (und über die relevante Software) ist in Donkers und Tuyls 2008 zu finden.

14.10 Übungen

1. Gegeben seien zwei unabhängige Eingaben für ein Klassifikationsproblem, d. h., es gilt $p(x_1, x_2|C) = p(x_1|C)\,p(x_2|C)$. Wie können wir $p(x_1|x_2, C)$ berechnen? Leiten Sie die Formel für $p(x_j|C_i) \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$ her.

2. Zeigen Sie, dass Gleichung 14.10 für einen Spitze-an-Spitze-Knoten auf $P(X, Y) = P(X) \cdot P(Y)$ führt.

   LÖSUNG: Wir wissen, dass $P(X, Y, Z) = P(Z|X, Y)\,P(X, Y)$, und wenn wir zudem wissen, dass $P(X, Y, Z) = P(X)\,P(Y)\,P(Z|X, Y)$, dann sehen wir, dass $P(X, Y) = P(X)\,P(Y)$ gelten muss.

3. Betrachten Sie noch einmal Abbildung 14.4 und berechnen Sie $P(R|W)$, $P(R|W, S)$ und $P(R|W, \sim S)$.

Lösung:

$$
\begin{aligned}
P(R|W) &= \frac{P(R,W)}{P(W)} = \frac{\sum_S P(R,W,S)}{\sum_R \sum_S P(R,W,S)} \\
&= \frac{\sum_S P(R)\,P(S)\,P(W|R,S)}{\sum_R \sum_S P(R)\,P(S)\,P(W|R,S)} \\
P(R|W,S) &= \frac{P(R,W,S)}{P(W,S)} = \frac{P(R)\,P(S)\,P(W|R,S)}{\sum_R P(R)\,P(S)\,P(W|R,S)} \\
P(R|W,\sim S) &= \frac{P(R,W,\sim S)}{P(W,\sim S)} = \frac{P(R)\,P(\sim S)\,P(W|R,\sim S)}{\sum_R P(R)\,P(\sim S)\,P(W|R,\sim S)}
\end{aligned}
$$

4. In Gleichung 14.30 ist X binär. Wie muss die Gleichung modifiziert werden, wenn X einen von K diskreten Werten annehmen kann?

   Lösung: Nehmen wir an, es gibt $j = 1, \ldots, K$ Zustände. Dann müssen wir, um das Modell linear zu halten, jeden Zustand mit einem separaten $\boldsymbol{w}_j$ parametrisieren und mit Softmax die Wahrscheinlichkeiten abbilden:

$$
P(X = j|U_1, \ldots, U_k, \{w_{ji}\}) = \frac{\exp \sum_{i=1}^{k} w_{ji} U_i + w_{j0}}{\sum_{l=1}^{K} \exp \sum_{i=1}^{k} w_{li} U_i + w_{l0}} \,.
$$

5. Zeigen Sie, dass für einen gerichteten Graphen, in dem die Verbundwahrscheinlichkeit wie in Gleichung 14.12 geschrieben werden kann, $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$ gilt.

   Lösung: Die Terme kürzen sich heraus, wenn wir über alle möglichen Werte summieren, weil diese Werte Wahrscheinlichkeiten sind. Betrachten wir zum Beispiel Abbildung 14.3:

$$
\begin{aligned}
P(X,Y,Z) &= P(X)\,P(Y|X)\,P(Z|X) \\
\sum_X \sum_Y \sum_Z P(X,Y,Z) &= \sum_X \sum_Y \sum_Z P(X)\,P(Y|X)\,P(Z|X) \\
&= \sum_X \sum_Y P(X)\,P(Y|X) \sum_Z P(Z|X) \\
&= \sum_X \sum_Y P(X)\,P(Y|X) \sum_Z \frac{P(Z,X)}{P(X)} \\
&= \sum_X \sum_Y P(X)\,P(Y|X) \frac{P(X)}{P(X)} \\
&= \sum_X \sum_Y P(X)\,P(Y|X) \\
&= \sum_X P(X) \sum_Y P(Y|X) = \sum_X P(X) = 1 \,.
\end{aligned}
$$

6. Zeichnen Sie den Necker-Würfel als Graphenmodell, indem Sie Verbindungen definieren, welche die gegenseitigen Beziehungen (Verstärkung oder Hemmung) zwischen den verschiedenen Interpretationen der Ecken anzeigen.

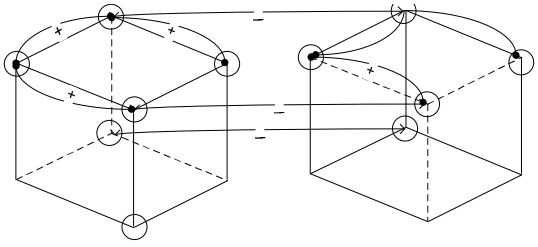


***Abb. 14.15*:** Zwei verschiedene Interpretationen des Necker-Würfels. Durchgezogene Linien, markiert durch „+“, sind anregend und gestrichelte Linien, markiert durch „-“ sind hemmend.

LÖSUNG: Es gibt Knoten, die den Ecken entsprechen, und die Werte, die sie annehmen, hängen von der Interpretation ab. Es gibt positive (verstärkende oder anregende) Verbindungen zwischen Ecken, die Teil der gleichen Interpretation sind, und es gibt negative (hemmende) Verbindungen zwischen Ecken, die zu unterschiedlichen Interpretationen gehören (siehe Abbildung 14.15).

7. Schreiben Sie nach dem Vorbild von Abbildung 14.7 ein Graphenmodell für die lineare logistische Regression für zwei Klassen auf.

8. Schlagen Sie ein geeignetes Gütemaß vor, das beim Lernen der Graphenstruktur für die Suche im Zustandsraum verwendet werden kann. Was sind geeignete Operatoren?

   LÖSUNG: Wir brauchen eine Score-Funktion, die sich additiv aus zwei Teilen zusammensetzt: einem, der die Güte der Anpassung quantifiziert (d. h. wie wahrscheinlich die Daten unter dem Modell sind), und einem, der die Komplexität des Graphen quantifiziert, um die Überanpassung zu verringern. Bei der Messung der Komplexität müssen wir die Gesamtzahl der Knoten berücksichtigen sowie die Anzahl der Parameter, die notwendig sind, um die bedingten Wahrscheinlichkeitsverteilungen zu repräsentieren. Beispielsweise sollten wir versuchen, Knoten mit so wenigen Eltern wie möglich zu haben. Mögliche Operatoren sind solche für das Hinzufügen/Entfernen einer Kante und für das Hinzufügen/Entfernen eines verborgenen Knotens.

9. Angenommen, in einer Zeitung schreibt ein Reporter an mehreren aufeinanderfolgenden Tagen eine Serie von Artikeln, die die gleichen Themen berühren, während sich die Story weiterentwickelt. Wie können wir dies mithilfe eines Graphenmodells modellieren?

14.11 Literaturangaben

Andrieu, C., N. de Freitas, A. Doucet, and M. I. Jordan. 2003. „An Introduction to MCMC for Machine Learning." *Machine Learning* 50: 5–43.

Barnard, K., P. Duygulu, D. Forsyth, N. de Freitas, D. M. Blei, and M. I. Jordan. 2003. „Matching Words and Pictures." *Journal of Machine Learning Research* 3: 1107–1135.

Bilmes, J., and C. Bartels. 2005. „Graphical Model Architectures for Speech Recognition." *IEEE Signal Processing Magazine* 22: 89–100.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning.* New York: Springer.

Buntine, W. 1996. „A Guide to the Literature on Learning Probabilistic Networks from Data." *IEEE Transactions on Knowledge and Data Engineering* 8: 195– 210.

Cowell, R. G., A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. 1999. *Probabilistic Networks and Expert Systems.* New York: Springer.

Donkers, J., and K. Tuyls. 2008. „Belief Networks in Bioinformatics." In *Computational Intelligence in Bioinformatics,* ed. A. Kelemen, A. Abraham, and Y. Chen, 75–111. Berlin: Springer.

Ghahramani, Z. 2001. „An Introduction to Hidden Markov Models and Bayesian Networks." *International Journal of Pattern Recognition and Artificial Intelligence* 15: 9–42.

Jensen, F. 1996. *An Introduction to Bayesian Networks.* New York: Springer.

Jordan, M. I., ed. 1999. *Learning in Graphical Models.* Cambridge, MA: MIT Press.

Jordan, M. I. 2004. „Graphical Models." *Statistical Science* 19: 140–155.

Jordan, M. I., Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. 1999. „An Introduc- tion to Variational Methods for Graphical Models." In *Learning in Graphical Models,* ed. M. I. Jordan, 105–161. Cambridge, MA: MIT Press.

Kschischang, F. R., B. J. Frey, and H.-A. Loeliger. 2001. „Factor Graphs and the Sum-Product Algorithm." *IEEE Transactions on Information Theory* 47: 498– 519.

Lauritzen, S. L., and D. J. Spiegelhalter. 1988. „Local Computations with Prob- abilities on Graphical Structures and their Application to Expert Systems." *Journal of Royal Statistical Society B* 50: 157–224.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms. Cambridge,* UK: Cambridge University Press.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective.* Cambridge, MA: MIT Press.

Neapolitan, R. E. 2004. *Learning Bayesian Networks.* Upper Saddle River, NJ: Pearson.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Francisco, CA: Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference.* Cambridge, UK: Cambridge University Press.

Sudderth, E. B., A. Torralba, W. T. Freeman, and A. S. Willsky. 2008. „Describing Visual Scenes Using Transformed Objects and Parts.“ *International Journal of Computer Vision* 77: 291–330.

Zweig, G. 2003. „Bayesian Network Structures and Inference Techniques for Automatic Speech Recognition.“ *Computer Speech and Language* 17: 173–193.

15 Hidden-Markov-Modelle

Wir schwächen die Vermutung ab, dass Instanzen in einer Stichprobe unabhängig sind, und führen Markov-Modelle ein, um Eingabesequenzen so zu modellieren, wie sie durch einen parametrischen Zufallsprozess generiert werden. Wir diskutieren, wie diese Modellierung durchgeführt wird und führen einen Algorithmus ein, mit dem die Parameter eines solchen Modells anhand gegebener Beispielsequenzen gelernt werden können.

15.1 Einführung

Bisher sind wir davon ausgegangen, dass die Instanzen, die eine Stichprobe ausmachen, unabhängig und identisch verteilt sind. Das hat den Vorteil, dass die Likelihood der Stichprobe schlicht das Produkt der Likelihood-Werte der individuellen Instanzen ist. Diese Annahme ist jedoch in Applikationen nicht haltbar, in denen sukzessive Instanzen voneinander abhängig sind. In einem Wort zum Beispiel sind aufeinanderfolgende Buchstaben abhängig; im Englischen folgt das „h“ mit großer Wahrscheinlichkeit dem „t“, jedoch nicht dem „x“. Solche Prozesse, bei denen es eine *Sequenz* von Beobachtungen gibt – zum Beispiel Buchstaben in einem Wort, Basenpaare in einer DNS-Sequenz – können nicht als einfache Wahrscheinlichkeitsverteilungen modelliert werden. Ein ähnliches Beispiel bildet die Spracherkennung, bei der sprachliche Äußerungen aus Sprachprimitiven, sogenannten Phonemen, zusammengesetzt sind; nur gewisse Sequenzen von Phonemen sind zulässig, nämlich diejenigen, welche die Worte der Sprache bilden. Auf einer höheren Ebene können Wörter in gewissen Sequenzen gesprochen oder geschrieben werden, um Sätze gemäß den syntaktischen und semantischen Regeln der Sprache zu formen.

Eine Sequenz kann dadurch charaktisiert werden, dass sie durch einen *parametrischen Zufallsprozess* generiert wurde. In diesem Kapitel befassen wir uns damit, wie diese Modellierung stattfindet und wie die Parameter solch eines Modells anhand einer Trainingsstichprobe mit Beispielsequenzen gelernt werden können.

15.2 Diskrete Markov-Prozesse

Stellen wir uns ein System vor, welches sich zu jedem beliebigen Zeitpunkt in einem in einer Menge N enthaltenen eindeutigen Zustand befindet: $S_1, S_2, \ldots, S_N$. Der Zustand zum Zeitpunkt t wird mit q_t, $t = 1, 2, \ldots$ bezeichnet, so dass beispielsweise $q_t = S_i$ bedeutet, dass sich das System zum Zeitpunkt t im Zustand S_i befindet. Zwar sprechen wir von der „Zeit“, als ob es sich um eine temporale Sequenz handelt, jedoch ist die Methodologie für jegliche Art Sequenz gültig, sei es eine zeitliche, eine räumliche, eine Position auf einem DNS-Strang oder irgendeine andere Sequenz.

Zu regelmäßig voneinander entfernten diskreten Zeitpunkten geht das System in einen Zustand mit einer gegebenen Wahrscheinlichkeit über, in Abhängigkeit von den Werten der vorherigen Zustände:

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \cdots) \,.$$

Markov-Modell

Für den Sonderfall eines *Markov-Modells* erster Ordnung hängt der Zustand zum Zeitpunkt $t + 1$ nur vom Zustand zum Zeitpunkt t ab, unabhängig von den Zuständen zu vorangegangen Zeitpunkten:

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \cdots) = P(q_{t+1} = S_j | q_t = S_i) \,. \quad (15.1)$$

Dies entspricht der Aussage, dass die Zukunft bei einem gegebenen gegenwärtigen Zustand unabhängig von der Vergangenheit ist. Dabei handelt es sich lediglich um die mathematische Version der Redewendung „Heute ist der erste Tag vom Rest deines Lebens“.

Übergangswahrscheinlichkeit

Wir vereinfachen das Modell weiter – d.h., wir regulieren es – indem wir annehmen, dass diese *Übergangswahrscheinlichkeiten* unabhängig von der Zeit sind:

$$a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i) \,, \quad (15.2)$$

wobei folgende Bedingungen erfüllt sind:

$$a_{ij} \geq 0 \text{ und } \sum_{j=1}^{N} a_{ij} = 1 \,. \quad (15.3)$$

Somit hat der Übergang von S_i zu S_j die gleiche Wahrscheinlichkeit, egal wann dies geschieht oder an welcher Stelle in der Beobachtungssequenz. $\mathbf{A} = [a_{ij}]$ ist eine $N \times N$ Matrix, deren Zeilen sich zu 1 summieren.

stochastischer Automat

Das Ganze kann als ein *stochastischer Automat* (siehe Abbildung 15.1) betrachtet werden. Von jedem Zustand S_i bewegt sich das System in einen

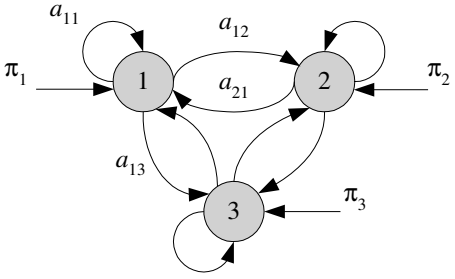


Abb. 15.1: Beispiel eines Markov-Modells mit drei Zuständen, was einem stochastischen Automaten gleichkommt. π_i ist die Wahrscheinlichkeit, dass das System im Zustand S_i beginnt und a_{ij} ist die Wahrscheinlichkeit, dass das System vom Zustand S_i in den Zustand S_j übergeht.

Zustand S_j mit der Wahrscheinlichkeit a_{ij}, und diese Wahrscheinlichkeit ist die gleiche für jeden beliebigen Zeitpunkt t. Der einzige Sonderfall ist der erste Zustand. Wir definieren *Ausgangswahrscheinlichkeiten* π_i, wobei es sich um die Wahrscheinlichkeit handelt, dass der erste Zustand in der Sequenz S_i ist:

AUSGANGSWAHRSCHEINLICHKEIT

$$\pi_i \equiv P(q_1 = S_i)\,, \tag{15.4}$$

wobei gilt:

$$\sum_{i=1}^{N} \pi_i = 1\,. \tag{15.5}$$

$\mathbf{\Pi} = [\pi_i]$ ist ein Vektor mit N Elementen, die sich zu 1 summieren.

Bei einem *beobachtbaren Markov-Modell* können die Zustände beobachtet werden. Zu jedem Zeitpunkt t kennen wir q_t, und während das System von einem Zustand in den nächsten übergeht, erhalten wir eine Beobachtungssequenz, das heißt eine Sequenz von Zuständen. Die Ausgabe des Prozesses ist die Menge an Zuständen zu jedem Zeitpunkt, wobei jeder Zustand einem physisch beobachtbaren Ereignis entspricht.

BEOBACHTBARES MARKOV-MODELL

Wir haben eine Beobachtungssequenz O, das heißt die Zustandssequenz $O = Q = \{q_1 q_2 \cdots q_T\}$, deren Wahrscheinlichkeit durch

$$P(O = Q|\mathbf{A}, \mathbf{\Pi}) = P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) = \pi_{q_1} a_{q_1 q_2} \cdots a_{q_{T-1} q_T} \tag{15.6}$$

gegeben ist.

π_{q_1} ist die Wahrscheinlichkeit dafür, dass der erste Zustand q_1 ist, $a_{q_1 q_2}$ ist die Wahrscheinlichkeit für den Übergang von q_1 zu q_2, und so weiter. Wir multiplizieren diese Wahrscheinlichkeiten, um die Wahrscheinlichkeit der gesamten Sequenz zu erhalten.

Betrachten wir nun einmal ein Beispiel (Rabiner und Juang 1986), um das Ganze zu verdeutlichen. Nehmen wir an, wir haben N Urnen und

jede Urne enthält Kugeln einer Farbe. Es gibt also eine Urne mit roten Kugeln, eine mit blauen Kugeln, und so weiter. Jemand zieht nun eine Kugel nach der anderen aus den Urnen und zeigt uns ihre Farbe. Sei q_t die Farbe der zum Zeitpunkt t gezogenen Kugel. Nehmen wir weiterhin an, es gibt drei Zustände:

$$S_1 : \text{rot}, S_2 : \text{blau}, S_3 : \text{grün}$$

mit den Anfangswahrscheinlichkeiten

$$\mathbf{\Pi} = [0,5, 0,2, 0,3]^T .$$

a_{ij} ist die Wahrscheinlichkeit dafür, dass aus Urne j (eine Kugel der Farbe j) gezogen wird, nachdem eine Kugel der Farbe i aus Urne i gezogen wurde. Die Übergangsmatrix ist zum Beispiel

$$\mathbf{A} = \begin{bmatrix} 0,4 & 0,3 & 0,3 \\ 0,2 & 0,6 & 0,2 \\ 0,1 & 0,1 & 0,8 \end{bmatrix} .$$

Bei gegebenem $\mathbf{\Pi}$ und $\mathbf{A}$ ist es einfach, K zufällige Sequenzen, jede mit der Länge T, zu generieren. Betrachten wir einmal, wie wir die Wahrscheinlichkeit einer Sequenz berechnen können. Angenommen, die ersten vier Kugeln sind „rot, rot, grün, grün“. Das entspricht der Beobachtungssequenz $O = \{S_1, S_1, S_3, S_3\}$. Ihre Wahrscheinlichkeit ergibt sich aus

$$\begin{aligned} P(O|\mathbf{A}, \mathbf{\Pi}) &= P(S_1) \cdot P(S_1|S_1) \cdot P(S_3|S_1) \cdot P(S_3|S_3) \\ &= \pi_1 \cdot a_{11} \cdot a_{13} \cdot a_{33} \\ &= 0,5 \cdot 0,4 \cdot 0,3 \cdot 0,8 = 0,048 . \end{aligned} \tag{15.7}$$

Sehen wir uns nun an, wie wir die Parameter, $\mathbf{\Pi}, \mathbf{A}$, lernen können. Gegeben seien K Sequenzen der Länge T, wobei q_t^k der Zustand zum Zeitpunkt t der Sequenz k ist; dann ist die Schätzung der Anfangswahrscheinlichkeit die Anzahl an Sequenzen, die mit S_i beginnen, geteilt durch die Gesamtzahl an Sequenzen:

$$\hat{\pi}_i = \frac{\#\{\text{Sequenzen beginnend mit } S_i\}}{\#\{\text{Anzahl an Sequenzen}\}} = \frac{\sum_k 1(q_1^k = S_i)}{K} , \tag{15.8}$$

wobei $1(b)$ gleich 1 ist, falls b wahr ist und andernfalls gleich 0.

Bezüglich der Übergangswahrscheinlichkeiten ist die Schätzung für a_{ij} die Anzahl an Übergängen von S_i nach S_j, geteilt durch die Gesamtzahl

an Übergängen von S_i über alle Sequenzen:

$$\begin{aligned}\hat{a}_{ij} &= \frac{\#\{\text{Übergänge von } S_i \text{ nach } S_j\}}{\#\{\text{Übergänge von } S_i\}} \\ &= \frac{\sum_k \sum_{t=1}^{T-1} 1(q_t^k = S_i \text{ und } q_{t+1}^k = S_j)}{\sum_k \sum_{t=1}^{T-1} 1(q_t^k = S_i)} \,. \end{aligned} \tag{15.9}$$

$\hat{a}_{12}$ ist die Anzahl, wie oft eine blaue Kugel einer roten folgt, geteilt durch die Gesamtzahl an Ziehungen roter Kugeln über alle Sequenzen.

15.3 Hidden-Markov-Modelle

Bei einem *Hidden-Markov-Modell* (HMM; dt. *verborgenes Markov-Modell*) sind die Zustände nicht beobachtbar; wenn wir jedoch einen Zustand erreichen, dann wird eine Beobachtung aufgezeichnet, d. h. eine probabilistische Funktion des Zustands. Wir gehen von einer diskreten Beobachtung in jedem Zustand aus der Menge $\{v_1, v_2, \ldots, v_M\}$ aus:

Hidden-Markov-Modell

$$b_j(m) \equiv P(O_t = v_m | q_t = S_j) \,. \tag{15.10}$$

$b_j(m)$ ist die *Beobachtungs-* oder *Emissionswahrscheinlichkeit* dafür, dass wir $v_m, m = 1, \ldots, M$ im Zustand S_j beobachten. Wir gehen erneut von einem homogenen Modell aus, in welchem die Wahrscheinlichkeiten nicht von t abhängen. Die so beobachteten Werte bilden die Beobachtungssequenz O. Die Zustandssequenz Q wird nicht beobachtet – denn das ist das „Verborgene“ am Modell – sollte jedoch anhand der Beobachtungssequenz O hergeleitet werden. Man beachte, dass es typischerweise viele verschiedene Zustandssequenzen Q gibt, welche dieselbe Beobachtunssequenz O generiert haben könnten, jedoch mit unterschiedlichen Wahrscheinlichkeiten. Da es bei einer gegebenen unabhängigen und identisch verteilten Stichprobe aus einer Normalverteilung eine unendliche Zahl an (μ, σ)-Wertepaaren geben kann, interessiert uns hier diejenige Sequenz, die am wahrscheinlichsten die Stichprobe generiert haben könnte.

Beobachtungswahrscheinlichkeit

Emissionswahrscheinlichkeit

Man beachte auch, dass es in diesem Fall eines Hidden-Markov-Modells zwei Quellen für die Zufälligkeit gibt: zusätzlich zu den zufälligen Übergängen von einem Zustand in einen anderen ist auch die Beobachtung in einem Zustand zufällig.

Betrachten wir noch einmal zu unserer Beispiel. Der verborgene Fall entspricht dem Beispiel mit Urnen und Kugeln, bei dem jede Urne Kugeln mehrerer Farben enthält. Sei $b_j(m)$ die Wahrscheinlichkeit dafür, dass eine Kugel der Farbe m aus der Urne j gezogen wird. Wir beobachten erneut eine Sequenz an Kugelfarben, jedoch ohne die Sequenz der Urnen zu kennen, aus denen die Kugeln gezogen wurden. Es ist also so, als

wären die Urnen diesmal hinter einem Vorhang platziert und jemand zieht zufällig eine Kugel aus einer der Urnen und zeigt uns nur diese, nicht aber die Urne, aus der sie stammt. Die Kugel wird in die Urne zurückgelegt, um die Wahrscheinlichkeiten nicht zu verändern. Die Anzahl an Kugelfarben muss nicht gleich der Anzahl an Urnen sein. Sagen wir zum Beispiel, wir haben drei Urnen und die Beobachtungssequenz sei

$$O = \{\text{rot, rot, grün, blau, gelb}\} .$$

Im vorherigen Fall kannten wir die Beobachtung (Kugelfarbe) und wussten somit auch genau über den Zustand (die Urne) Bescheid, da separate Urnen für die unterschiedlichen Farben existierten und jede Urne nur Kugeln derselben Farbe enthielt. Das beobachtbare Markov-Modell ist ein Sonderfall des Hidden-Markov-Modells, bei dem $M = N$ und $b_j(m)$ gleich 1 ist, falls $j = m$ und in allen andere Fällen gleich 0. Im Falle eines verborgenen Modells aber hätte eine Kugel aus jeder beliebigen Urne gezogen worden sein können. In diesem Fall gibt es für dieselbe Beobachtungssequenz O viele mögliche Zustandssequenzen Q, die O hätten generieren können (siehe Abbildung 15.2).

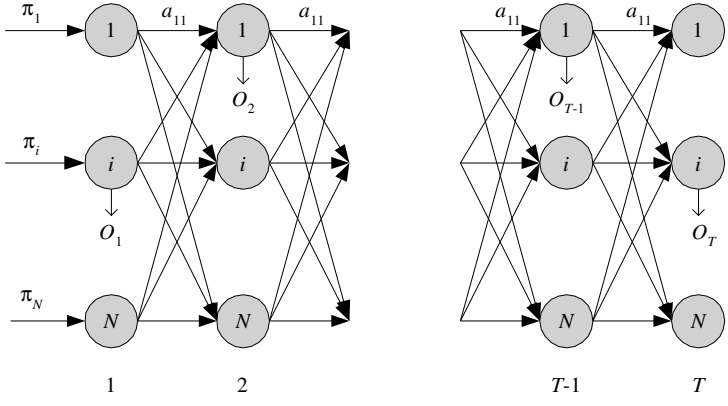


***Abb. 15.2*:** Ein zeitlich als Verband (oder Gitter) entfaltetes HMM, welches alle möglichen Zustandsfolgen zeigt. Einer der Pfade – dargestellt durch dickere Linien – ist die tatsächliche (unbekannte) Zustandsfolge, welche die Beobachtungssequenz generiert hat.

Fassen wir noch einmal zusammen und formulieren wir die Elemente eines HMM wie folgt:

1. N: Anzahl an Zuständen des Modells

   $$S = \{S_1, S_2, \ldots, S_N\}$$

2. M: Anzahl an eindeutigen Beobachtungssymbolen im *Alphabet*

   $$V = \{v_1, v_2, \ldots, v_M\}$$

3. Zustandsübergangswahrscheinlichkeiten:

$$\mathbf{A} = [a_{ij}] \text{ mit } a_{ij} \equiv P(q_{t+1} = S_j | q_t = S_i)$$

4. Beobachtungswahrscheinlichkeiten:

$$\mathbf{B} = [b_j(m)] \text{ mit } b_j(m) \equiv P(O_t = v_m | q_t = S_j)$$

5. Anfangszustandswahrscheinlichkeiten:

$$\mathbf{\Pi} = [\pi_i] \text{ mit } \pi_i \equiv P(q_1 = S_i)\,.$$

N und M sind implizit in den anderen Parametern definiert, so dass $\lambda = (\mathbf{A}, \mathbf{B}, \mathbf{\Pi})$ die Parametermenge eines HMM bezeichnet. Ist λ gegeben, so kann das Modell genutzt werden, um eine willkürliche Anzahl an Beobachtungssequenzen von willkürlicher Länge zu generieren – wie üblich jedoch interessiert uns genau das Entgegengesetzte, nämlich die Schätzung der Parameter des Modells bei vorliegendem Trainingsdatensatz mit Sequenzen.

15.4 Drei grundsätzliche Probleme eines HMM

Bei einer vorliegenden Sequenz von Beobachtungen interessieren uns drei Problemstellungen:

1. Bei vorliegendem Modell λ, möchten wir die Wahrscheinlichkeit einer beliebigen Beobachtungssequenz, $O = \{O_1 O_2 \cdots O_T\}$, nämlich $P(O|\lambda)$, berechnen.

2. Bei vorliegendem Modell λ und einer Beobachtungssequenz O wollen wir die Zustandssequenz $Q = \{q_1 q_2 \cdots q_T\}$ herausfinden, welche am wahrscheinlichsten O generiert haben könnte, sprich wir wollen Q^* finden, welches $P(Q|O, \lambda)$ maximiert.

3. Bei vorliegendem Trainingsdatensatz mit Beobachtungssequenzen, $\mathcal{X} = \{O^k\}_k$, wollen wir das Modell lernen, welches die Wahrscheinlichkeit $\mathcal{X}$ zu generieren maximiert, das heißt, wir wollen λ^* finden, welches $P(\mathcal{X}|\lambda)$ maximiert.

Lassen Sie uns die Lösungen hierzu eine nach der anderen betrachten, wobei jede Lösung genutzt wird, um das nächste Problem anzugehen, bis wir bei der Berechnung von λ oder dem Lernen eines Modells anhand der Daten angekommen sind.

15.5 Evaluierungsproblem

Gegeben sei eine Beobachtungssequenz $O = \{O_1 O_2 \cdots O_T\}$ und eine Zustandssequenz $Q = \{q_1 q_2 \cdots q_T\}$; dann ist die Wahrscheinlichkeit O unter gegebener Zustandssequenz Q zu beobachten einfach

$$P(O|Q,\lambda) = \prod_{t=1}^{T} P(O_t|q_t,\lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T)\,, \tag{15.11}$$

was wir nicht berechnen können, da wir die Zustandssequenz nicht kennen. Die Wahrscheinlichkeit der Zustandssequenz Q ist

$$P(Q|\lambda) = P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) = \pi_{q_1} a_{q_1 q_2} \cdots a_{q_{T-1} q_T}\,. \tag{15.12}$$

Dann ist die Verbundwahrscheinlichkeit gleich

$$\begin{aligned} P(O,Q|\lambda) &= P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}) \prod_{t=1}^{T} P(O_t|q_t) \\ &= \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T)\,. \end{aligned} \tag{15.13}$$

Wir können $P(O|\lambda)$ berechnen, indem wir über den Verbund marginalisieren, sprich, indem wir über alle möglichen Q summieren:

$$P(O|\lambda) = \sum_{\text{alle möglichen } Q} P(O,Q|\lambda)\,.$$

Vorwärts-Rückwärts-Prozedur

Jedoch ist dies nicht sehr praktisch, da es N^T mögliche Q gibt, wenn angenommen wird, dass alle Wahrscheinlichkeiten ungleich 0 sind. Glücklicherweise existiert eine effiziente Prozedur zur Berechnung von $P(O|\lambda)$, welche man als *Vorwärts-Rückwärts-Prozedur* bezeichnet. Sie basiert auf der Idee, die Beobachtungssequenz in zwei Teile zu zerlegen: der erste Teil der Prozedur läuft vom Zeitpunkt 1 bis zum Zeitpunkt t und der zweite erstreckt sich vom Zeitpunkt $t+1$ bis zu T.

Vorwärtsvariable

Wir definieren die *Vorwärtsvariable* $\alpha_t(i)$ als die Wahrscheinlichkeit für die Beobachtung der Teilsequenz $\{O_1 \cdots O_t\}$ bis zum Zeitpunkt t und dafür, dass zum Zeitpunkt t der Zustand S_i vorliegt, bei gegebenem Modell λ:

$$\alpha_t(i) \equiv P(O_1 \cdots O_t, q_t = S_i|\lambda)\,. \tag{15.14}$$

Das Nützliche daran ist die Möglichkeit zur rekursiven Berechnung durch die Akkumulation der Ergebnisse entlang des Weges:

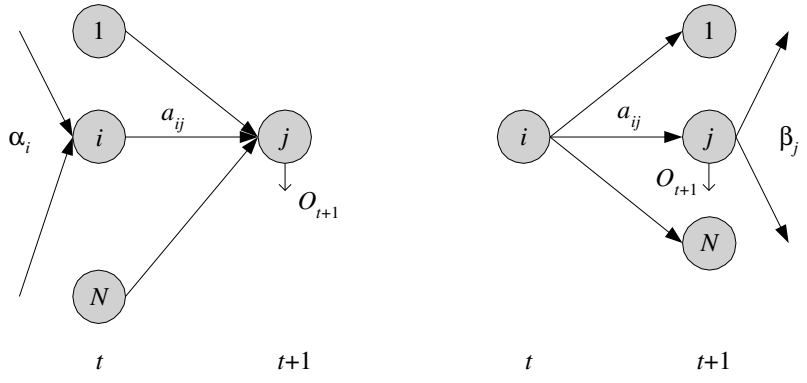
- Initialisierung:

$$\begin{aligned}\alpha_1(i) &\equiv P(O_1, q_1 = S_i|\lambda) \\ &= P(O_1|q_1 = S_i, \lambda)P(q_1 = S_i|\lambda) \\ &= \pi_i b_i(O_1)\end{aligned} \tag{15.15}$$

- Rekursion (siehe Abbildung 15.3a):

$$\begin{aligned}&\alpha_{t+1}(j) \equiv P(O_1 \cdots O_{t+1}, q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_{t+1}|q_{t+1} = S_j, \lambda)P(q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_t|q_{t+1} = S_j, \lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda)P(q_{t+1} = S_j|\lambda) \\ &= P(O_1 \cdots O_t, q_{t+1} = S_j|\lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \sum_i P(O_1 \cdots O_t, q_t = S_i, q_{t+1} = S_j|\lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &\quad \times \sum_i P(O_1 \cdots O_t, q_{t+1} = S_j|q_t = S_i, \lambda)P(q_t = S_i|\lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &\sum_i P(O_1 \cdots O_t|q_t = S_i, \lambda)P(q_{t+1} = S_j|q_t = S_i, \lambda)P(q_t = S_i|\lambda) \\ &= P(O_{t+1}|q_{t+1} = S_j, \lambda) \\ &\quad \times \sum_i P(O_1 \cdots O_t, q_t = S_i|\lambda)P(q_{t+1} = S_j|q_t = S_i, \lambda) \\ &= \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1})\,.\end{aligned} \tag{15.16}$$

$\alpha_t(i)$ erklärt die ersten t Beobachtungen und endet im Zustand S_i. Wir multiplizieren dies mit der Wahrscheinlichkeit a_{ij} für den Übergang in den Zustand S_j, und weil es N mögliche vorherige Zustände gibt, müssen wir über alle solchen möglichen vorhergehenden S_i summieren. $b_j(O_{t+1})$ ist dann die Wahrscheinlichkeit, dass wir die $(t+1)$-te Beobachtung generieren, während wir uns im Zustand S_j zum Zeitpunkt $t+1$ befinden.



(a) Vorwärts (b) Rückwärts

***Abb. 15.3*:** Vorwärts-Rückwärts-Prozedur: (a) Berechnung von $\alpha_t(j)$ und (b) Berechnung von $\beta_t(i)$.

Wenn wir die Vorwärtsvariablen berechnen, ist es ein Leichtes, die Wahrscheinlichkeit der Beobachtungssequenz zu bestimmen:

$$\begin{aligned} P(O|\lambda) &= \sum_{i=1}^{N} P(O, q_T = S_i|\lambda) \\ &= \sum_{i=1}^{N} \alpha_T(i) \,. \end{aligned} \tag{15.17}$$

$\alpha_T(i)$ ist die Wahrscheinlichkeit, dass die gesamte Beobachtungssequenz generiert wird und der letzte Zustand S_i ist. Wir müssen über alle solchen möglichen finalen Zustände summieren.

Die Berechung von $\alpha_t(i)$ ist $\mathcal{O}(N^2T)$, und damit ist unser erstes Evaluierungsproblem in annehmbarer Zeit lösbar. Wir benötigen sie zwar jetzt noch nicht, jedoch wollen wir auf ähnliche Weise die *Rückwärtsvariable* $\beta_t(i)$ definieren, welche der Wahrscheinlichkeit entspricht, im Zustand S_i zum Zeitpunkt t zu sein und die Teilsequenz $O_{t+1} \cdots O_T$ zu beobachten:

RÜCKWÄRTS-VARIABLE

$$\beta_t(i) \equiv P(O_{t+1} \cdots O_T|q_t = S_i, \lambda) \,. \tag{15.18}$$

Erneut kann dies wie folgt rekursiv berechnet werden, wobei diesmal rückwärts gegangen wird:

- Initialisierung (willkürlich auf 1):

  $$\beta_T(i) = 1$$

- Rekursion (siehe Abbildung 15.3b):

$$\begin{aligned}
\beta_t(i) &\equiv P(O_{t+1}\cdots O_T|q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1}\cdots O_T, q_{t+1} = S_j|q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1}\cdots O_T|q_{t+1} = S_j, q_t = S_i, \lambda) \\
&\quad \times P(q_{t+1} = S_j|q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1}|q_{t+1} = S_j, q_t = S_i, \lambda) \\
&\quad \times P(O_{t+2}\cdots O_T|q_{t+1} = S_j, q_t = S_i, \lambda) \\
&\quad \times P(q_{t+1} = S_j|q_t = S_i, \lambda) \\
&= \sum_j P(O_{t+1}|q_{t+1} = S_j, \lambda) P(O_{t+2}\cdots O_T|q_{t+1} = S_j, \lambda) \\
&\quad \times P(q_{t+1} = S_j|q_t = S_i, \lambda) \\
&= \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \,. \qquad (15.19)
\end{aligned}$$

Befinden wir uns im Zustand S_i, dann können wir in N mögliche nächste Zustände S_j übergehen, von denen jeder die Wahrscheinlichkeit a_{ij} besitzt. Während wir uns in diesem Zustand befinden, generieren wir die $(t+1)$-te Beobachtung und $\beta_{t+1}(j)$ erklärt von da an fortsetzend all die Beobachtungen nach dem Zeitpunkt $t+1$.

Ein Hinweis zur Vorsicht bezüglich der Implementierung sei hier gegeben: sowohl die Werte für α_t als auch für β_t werden berechnet, indem kleine Wahrscheinlichkeiten multipliziert werden. Bei langen Sequenzen besteht die Gefahr, dass die numerische Genauigkeit des Rechners nicht ausreicht, um solche kleinen Zahlen darzustellen. Um das zu vermeiden, normalisieren wir $\alpha_t(i)$ bei jedem Zeitschritt, indem wir mit

$$c_t = \frac{1}{\sum_j \alpha_t(j)}$$

multiplizieren.

Außerdem normalisieren wir $\beta_t(i)$, indem wir durch dasselbe c_t teilen ($\beta_t(i)$ summieren sich nicht zu 1). Gleichung 15.17 können wir nach der Normalisierung nicht verwenden, stattdessen haben wir (Rabiner 1989)

$$P(O|\lambda) = \frac{1}{\prod_t c_t} \quad \text{oder} \quad \log P(O|\lambda) = -\sum_t \log c_t \,. \qquad (15.20)$$

15.6 Herausfinden der Zustandssequenz

Wir widmen uns nun dem zweiten Problem, dem Herausfinden der Zustandssequenz $Q = \{q_1 q_2 \cdots q_T\}$, welche die höchste Wahrscheinlichkeit besitzt, bei gegebenem Modell λ die Beobachtungssequenz $O = \{O_1 O_2 \cdots O_T\}$ zu generieren.

Definieren wir $\gamma_t(i)$ als die Wahrscheinlichkeit dafür, dass wir uns zum Zeitpunkt t im Zustand S_i befinden, bei gegebenem O und λ, was wie folgt berechnet werden kann:

$$\begin{aligned}
\gamma_t(i) &\equiv P(q_t = S_i|O,\lambda) && (15.21)\\
&= \frac{P(O|q_t = S_i,\lambda)P(q_t = S_i|\lambda)}{P(O|\lambda)}\\
&= \frac{P(O_1 \cdots O_t|q_t = S_i,\lambda)P(O_{t+1} \cdots O_T|q_t = S_i,\lambda)P(q_t = S_i|\lambda)}{\sum_{j=1}^{N} P(O, q_t = S_j|\lambda)}\\
&= \frac{P(O_1 \cdots O_t, q_t = S_i|\lambda)P(O_{t+1} \cdots O_T|q_t = S_i,\lambda)}{\sum_{j=1}^{N} P(O|q_t = S_j,\lambda)P(q_t = S_j|\lambda)}\\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\,. && (15.22)
\end{aligned}$$

Hier erkennen wir, wie günstig $\alpha_t(i)$ und $\beta_t(i)$ die Sequenz untereinander aufteilen: die Vorwärtsvariable $\alpha_t(i)$ erklärt den Anfangsteil der Sequenz bis zum Zeitpunkt t und endet in S_i, und die Rückwärtsvariable $\beta_t(i)$ setzt an dieser Stelle fort und erklärt den Schlussteil bis zum Zeitpunkt T.

Der Zähler $\alpha_t(i)\beta_t(i)$ erklärt die komplette Sequenz, wenn das System zum Zeitpunkt t im Zustand S_i ist. Um zu normalisieren, dividieren wir durch die Summe dieses Ausrucks über alle möglichen Zwischenzustände, die zum Zeitpunkt t traversiert werden können. Dadurch wird garantiert, dass $\sum_i \gamma_t(i) = 1$ ist.

Um die Zustandssequenz herauszufinden, können wir für jeden Zeitpunkt t den Zustand wählen, der die höchste Wahrscheinlichkeit besitzt:

$$q_t^* = \operatorname*{argmax}_i \gamma_t(i)\,, \tag{15.23}$$

jedoch könnten dadurch S_i und S_j als die wahrscheinlichsten Zustände zum Zeitpunkt t und $t+1$ gewählt werden, selbst wenn $a_{ij} = 0$. Um eine einzelne beste *Zustandssequenz* (bzw. Pfad) herauszufinden, nutzen wir den *Viterbi-Algorithmus*, welcher auf der dynamischen Programmierung basiert und solche Übergangswahrscheinlichkeiten einbezieht.

Viterbi-Algorithmus

Bei gegebener Zustandssequenz $Q = q_1 q_2 \cdots q_T$ und Beobachtunssequenz $O = O_1 \cdots O_T$ definieren wir $\delta_t(i)$ als die Wahrscheinlichkeit des Pfades

mit der höchsten Wahrscheinlichkeit zum Zeitpunkt t, der für die ersten t Beobachtungen verantwortlich ist und in S_i endet:

$$\delta_t(i) \equiv \max_{q_1 q_2 \cdots q_{t-1}} p(q_1 q_2 \cdots q_{t-1}, q_t = S_i, O_1 \cdots O_t | \lambda) \,. \tag{15.24}$$

Damit kann $\delta_{t+1}(i)$ rekursiv berechnet werden. Den optimalen Pfad erhalten wir, wenn wir vom Zeitpunkt T schrittweise zurückgehen und zu jedem Zeitpunkt den wahrscheinlichsten Zustand mit größtem $\gamma_t(i)$ wählen. Der Algorithmus lautet wie folgt:

1. Initialisierung:

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(O_1) \\ \psi_1(i) &= 0 \end{aligned}$$

2. Rekursion:

$$\begin{aligned} \delta_t(j) &= \max_i \delta_{t-1}(i) a_{ij} \cdot b_j(O_t) \\ \psi_t(j) &= \operatorname*{argmax}_i \delta_{t-1}(i) a_{ij} \end{aligned}$$

3. Terminierung:

$$\begin{aligned} p^* &= \max_i \delta_T(i) \\ q_T^* &= \operatorname*{argmax}_i \delta_T(i) \end{aligned}$$

4. Zurückverfolgung des Pfads (der Zustandssequenz):

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \; t = T-1, T-2, \ldots, 1 \,.$$

Der Abbildung 15.2 folgend wird in $\psi_t(j)$ der Index des besten vorherigen Zustandes zum Zeitpunkt $t-1$ gespeichert, der ein maximales $\delta_t(j)$ ermöglicht. Der Viterbi-Algorithmus weist die gleiche Komplexität auf wie die Vorwärtsphase der oben beschriebenen Prozedur, wobei wir bei jedem Schritt statt der Summe das Maximum wählen.

15.7 Lernen von Modellparametern

Wir widmen uns nun dem dritten Problem, also dem Lernen eines HMM anhand von Daten. Dem Maximum-Likelihood-Ansatz folgend wollen wir λ^* berechnen, welches die Likelihood der Stichprobe von Trainingssequenzen, $\mathcal{X} = \{O^k\}_{k=1}^K$, maximiert, sprich $P(\mathcal{X}|\lambda)$. Wir beginnen mit

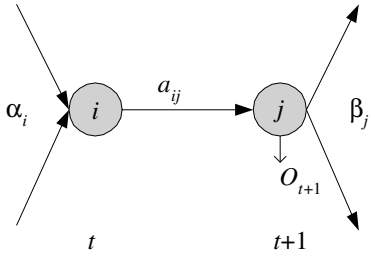


Abb. 15.4: Berechnung der Wahrscheinlichkeiten $\xi_t(i,j)$.

der Definition einer neuen Variable, die sich später als nützlich erweisen wird.

Wir definieren $\xi_t(i,j)$ bei Vorliegen der kompletten Beobachtung O und λ als die Wahrscheinlichkeit, sich zum Zeitpunkt t im Zustand S_i und zum Zeitpunkt $t+1$ im Zustand S_j zu befinden:

$$\xi_t(i,j) \equiv P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \,, \tag{15.25}$$

was wie folgt berechnet werden kann (siehe Abbildung 15.4):

$$\begin{aligned}
\xi_t(i,j) &\equiv P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \\
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda) P(q_t = S_i, q_{t+1} = S_j|\lambda)}{P(O|\lambda)} \\
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda) P(q_{t+1} = S_j | q_t = S_i, \lambda) P(q_t = S_i|\lambda)}{P(O|\lambda)} \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t | q_t = S_i, \lambda) P(O_{t+1} | q_{t+1} = S_j, \lambda) \\
&\quad \times P(O_{t+2} \cdots O_T | q_{t+1} = S_j, \lambda) a_{ij} P(q_t = S_i|\lambda) \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t, q_t = S_i|\lambda) P(O_{t+1} | q_{t+1} = S_j, \lambda) \\
&\quad \times P(O_{t+2} \cdots O_T | q_{t+1} = S_j, \lambda) a_{ij} \\
&= \frac{\alpha_t(i) b_j(O_{t+1}) \beta_{t+1}(j) a_{ij}}{\sum_k \sum_l P(q_t = S_k, q_{t+1} = S_l, O|\lambda)} \\
&= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_k \sum_l \alpha_t(k) a_{kl} b_l(O_{t+1}) \beta_{t+1}(l)} \,.
\end{aligned} \tag{15.26}$$

$\alpha_t(i)$ erklärt die ersten t Beobachtungen und endet zum Zeitpunkt t im Zustand S_i. Wir gehen in den Zustand S_j mit der Wahrscheinlichkeit a_{ij} über, generieren die $(t+1)$-te Beobachtung, und setzen von S_j an zum Zeitpunkt $t+1$ fort, um den Rest der Beobachtungssequenz zu generieren. Wir normalisieren, indem wir für alle derartigen Paare teilen, die zum Zeitpunkt t und $t+1$ besucht werden können.

Wenn wir wollen, können wir auch die Wahrscheinlichkeit dafür berechnen, dass wir uns zum Zeitpunkt t im Zustand S_i befinden, indem wir

über die Wahrscheinlichkeiten für alle möglichen nachfolgenden Zustände marginalisieren:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \,. \tag{15.27}$$

Man beachte, dass im Fall eines beobachtbaren statt eines Hidden-Markov-Modells sowohl $\gamma_t(i)$ als auch $\xi_t(i,j)$ gleich 0/1 wären. In diesem Fall, bei dem sie es nicht sind, schätzen wir sie mit a-posteriori-Wahrscheinlichkeiten, die uns *weiche Zuordnungen* liefern. Das entspricht dem Unterschied zwischen überwachter Klassifikation und einer unüberwachten Clustermethode, wo wir die Klassenzuordnungen kannten beziehungsweise nicht kannten. Bei der unüberwachten Clustermethode unter Verwendung des EM-Algorithmus (Abschnitt 7.4) schätzten wir zuerst die Klassenzuordnungen, ohne sie zu kennen (im E-Schritt) und berechneten dann die Parameter mit diesen Schätzungen (im M-Schritt).

WEICHE ZUORDUNGEN

Auf ähnliche Weise haben wir hier den *Baum-Welch-Algorithmus*, welcher eine EM-Prozedur ist. Bei jeder Iteration berechnen wir zuerst im E-Schritt die Werte $\xi_t(i,j)$ und $\gamma_t(i)$ für das gegenwärtige $\lambda = (\mathbf{A}, \mathbf{B}, \mathbf{\Pi})$, und berechnen dann im M-Schritt λ erneut bei gegebenem $\xi_t(i,j)$ und $\gamma_t(i)$. Diese zwei Schritte werden bis zur Konvergenz alterniert, während dabei, wie bereits bewiesen wurde, $P(O|\lambda)$ nie kleiner wird.

BAUM-WELCH-ALGORITHMUS

Es seien die Indikatorvariablen z_i^t angenommen als

$$z_i^t = \begin{cases} 1 \text{ falls } q_t = S_i \,, \\ 0 \text{ andernfalls} \end{cases} \tag{15.28}$$

und

$$z_{ij}^t = \begin{cases} 1 \text{ falls } q_t = S_i \text{ und } q_{t+1} = S_j \,, \\ 0 \text{ andernfalls} \,. \end{cases} \tag{15.29}$$

Sie sind 0/1 im Fall des beobachtbaren Markov-Modells und verborgene Zufallsvariablen im Fall des HMM. In diesem letzteren Fall schätzen wir sie im E-Schritt als

$$\begin{aligned} E[z_i^t] &= \gamma_t(i) \,, \\ E[z_{ij}^t] &= \xi_t(i,j) \,. \end{aligned} \tag{15.30}$$

Im M-Schritt berechnen wir die Parameter für diese gegebenen Schätzwerte. Die zu erwartende Anzahl an Übergängen von S_i nach S_j ist $\sum_t \xi_t(i,j)$, und die Gesamtzahl an Übergängen von S_i ist $\sum_t \gamma_t(i)$. Das Verhältnis dieser zwei liefert uns die Wahrscheinlichkeit des Übergangs von S_i nach S_j zu einem beliebigen Zeitpunkt:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \,. \tag{15.31}$$

Man beachte, dass es sich hierbei um Gleichung 15.9 handelt, nur dass die tatsächlichen Zählvariablen durch geschätzte weiche Zählvariablen ersetzt wurden.

Die Wahrscheinlichkeit v_m zu beobachten, während wir uns in S_j befinden, ist die zu erwartende Anzahl dafür, wie oft v_m beobachtet wird, wenn das System im Zustand S_j ist, gegenüber der gesamten Anzahl dafür, wie oft das System sich in S_j befindet:

$$\hat{b}_j(m) = \frac{\sum_{t=1}^{T} \gamma_t(j) 1(O_t = v_m)}{\sum_{t=1}^{T} \gamma_t(j)} \,. \tag{15.32}$$

Wenn multiple Beobachtungssequenzen existieren,

$$\mathcal{X} = \{O^k\}_{k=1}^{K} \,,$$

die wir als unabhängig annehmen,

$$P(\mathcal{X}|\lambda) = \prod_{k=1}^{K} P(O^k|\lambda) \,,$$

so sind die Parameter nun Durchschnittswerte über alle Beobachtungen in allen Sequenzen:

$$\begin{aligned} \hat{a}_{ij} &= \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \xi_t^k(i,j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \gamma_t^k(i)} \,, \\ \hat{b}_j(m) &= \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^k(j) 1(O_t^k = v_m)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^k(j)} \,, \\ \hat{\pi}_i &= \frac{\sum_{k=1}^{K} \gamma_1^k(i)}{K} \,. \end{aligned} \tag{15.33}$$

15.8 Kontinuierliche Beobachtungen

In unseren Betrachtungen sind wir von diskreten Beobachtungen ausgegangen, die auf multinomiale Weise modelliert wurden:

$$P(O_t|q_t = S_j, \lambda) = \prod_{m=1}^{M} b_j(m)^{r_m^t} \tag{15.34}$$

mit

$$r_m^t = \begin{cases} 1 \text{ falls } O_t = v_m \,, \\ 0 \text{ andernfalls} \,. \end{cases} \tag{15.35}$$

Sind die Eingaben kontinuierlich, dann besteht eine Möglichkeit darin, sie zu diskretisieren und dann diese diskreten Werte als Beobachtungen zu nutzen. Typischerweise wird ein Vektorquantisierer (Abschnitt 7.3) zu Zwecken der Konvertierung von kontinuierlichen Werten in den diskreten Index des nächstliegenden Referenzvektors verwendet. Bei der Spracherkennung beispielsweise wird eine Wortäußerung in kurze Sprachsegmente zerlegt, die Phonemen oder Teilen von Phonemen entsprechen; nach der Vorbearbeitung werden diese mit Hilfe eines Vektorquantisierers diskretisiert und dann wird ein HMM genutzt, um eine Wortäußerung als eine Sequenz aus ihnen zu modellieren.

Wir erinnern uns, dass der für die Vektorquantisierung genutzte k-Means die harte Version eines Modells gemischter Gaußverteilungen ist:

$$p(O_t|q_t = S_j, \lambda) = \sum_{l=1}^{L} P(\mathcal{G}_l) p(O_t|q_t = S_j, \mathcal{G}_l, \lambda) \tag{15.36}$$

mit

$$p(O_t|q_t = S_j, \mathcal{G}_l, \lambda) \sim \mathcal{N}(\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \tag{15.37}$$

und kontinuierlich gehaltenen Beobachtungen. Im diesem Fall von gemischten Gaußverteilungen können EM-Gleichungen für die Komponentenparameter (mit angemessener Regularisierung, um die Anzahl an Parametern in Grenzen zu halten) und die Proportionen der Mischung abgeleitet werden (Rabiner 1989).

Betrachten wir nun den Fall einer skalaren kontinuierlichen Beobachtung $O_t \in \mathbb{R}$. Am einfachsten ist es, von einer Normalverteilung auszugehen:

$$p(O_t|q_t = S_j, \lambda) \sim \mathcal{N}(\mu_j, \sigma_j^2)\,, \tag{15.38}$$

was impliziert, dass im Zustand S_j die Beobachtung aus einer Normalverteilung mit Mittelwert μ_j und Varianz σ_j^2 gezogen wird. Die Gleichungen des M-Schritts in diesem Fall sind

$$\begin{aligned} \hat{\mu}_j &= \frac{\sum_t \gamma_t(j) O_t}{\sum_t \gamma_t(j)}\,, \\ \hat{\sigma}_j^2 &= \frac{\sum_t \gamma_t(j)(O_t - \hat{\mu}_j)^2}{\sum_t \gamma_t(j)}\,. \end{aligned} \tag{15.39}$$

15.9 Das HMM als Graphenmodell

In Kapitel 14 haben wir Graphenmodelle diskutiert, und auch das Hidden-Markov-Modell kann durch ein solches dargestellt werden. Die drei aufeinanderfolgenden Zustände q_{t-2}, q_{t-1}, q_t entsprechen den drei Zuständen auf einer Kette in einem Markov-Modell erster Ordnung. Der Zustand

zur Zeit t, q_t, hängt nur vom Zustand zur Zeit $t-1$, q_{t-1}, ab und ist bei gegebenem q_{t-1} unabhängig von q_{t-2}:

$$P(q_t|q_{t-1}, q_{t-2}) = P(q_t|q_{t-1}) \,.$$

Die Übergangswahrscheinlichkeiten sind durch die Matrix $\mathbf{A}$ gegeben (siehe Abbildung 15.5). Jede verborgene Variable generiert eine Beobachtung, wobei die Beobachtungswahrscheinlichkeiten durch die Matrix $\mathbf{B}$ gegeben sind. Die in diesem Kapitel diskutierte Vorwärts-Rückwärts-Prozedur für HMMs ist eine Anwendung der Belief-Propagation, die wir in Abschnitt 14.5 kennengelernt haben.

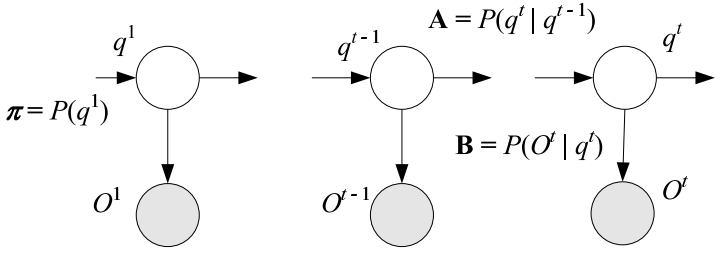


***Abb. 15.5*:** Ein Hidden-Markov-Modell kann als Graphenmodell dargestellt werden, in dem die q^t die verborgenen Zustände sind und die grau gezeichneten O^t beobachtet werden.

Eingabe-Ausgabe-HMM

Im Rahmen des Graphenformalismus lassen sich unterschiedliche HMM-Typen konstruieren, die jeweils durch ein eigenes Graphenmodell repräsentiert werden. In Abbildung 15.6a ist ein *Eingabe-Ausgabe-HMM* dargestellt, bei dem es zwei getrennte beobachtete Eingabe-Ausgabe-Sequenzen sowie eine Sequenz von verborgenen Zuständen gibt (Bengio und Frasconi 1996). Es gibt einige Anwendungen, bei denen das der Fall ist, d. h., zusätzlich zu der Sequenz von Beobachtungen O_t haben wir eine Sequenz von Eingaben x^t und wir schreiben $P(O_t|q_t = S_j, x_t)$. Wenn die Beobachtungen kontinuierliche Größen sind, ersetzen wir Gleichung 15.38 durch ein generalisiertes Modell

$$p(O_t|q_t = S_j, x_t, \lambda) \sim \mathcal{N}(g_j(x^t|\theta_j), \sigma_j^2) \,, \tag{15.40}$$

wobei wir zum Beispiel unter Annahme eines linearen Modells Folgendes erhalten:

$$g_j(x^t|w_j, w_{j0}) = w_j x^t + w_{j0} \,. \tag{15.41}$$

Wenn die Beobachtungen diskret und multinomial sind, dann haben wir einen Klassifikator, der x^t als Eingabe nutzt und eine 1-aus-M-Ausgabe generiert; oder wir können a-posteriori-Klassenwahrscheinlichkeiten generieren und die Beobachtungen kontinuierlich belassen.

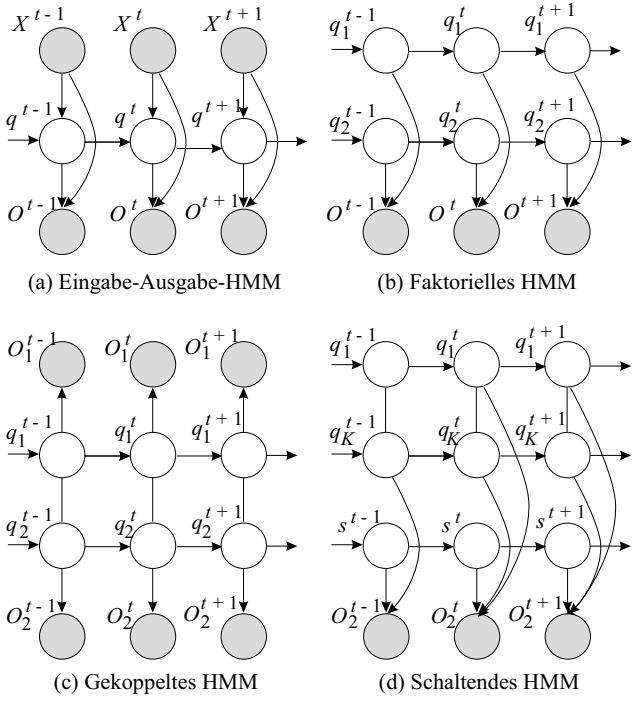


***Abb. 15.6*:** Unterschiedliche Typen von HMMs modellieren unterschiedliche Annahmen über die Art, wie die beobachteten Daten (grau) aus den Markovschen Sequenzen von verborgenen Variablen generiert werden.

Entsprechend können auch die Zustandsübergangswahrscheinlichkeiten von der Eingabe abhängig gemacht werden, also $P(q_{t+1} = S_j | q_t = S_i, x_t)$, was durch einen Klassifikator implementiert wird, der den Zustand zum Zeitpunkt $t + 1$ als Funktion des Zustands zum Zeitpunkt t und der Eingabe wählt. Hierbei spricht man von *Markovschen gemischten Expertensystemen* (Meila und Jordan 1996) und es handelt sich um eine Generalisierung der Architektur von gemischten Expertensystemen (Abschnitt 12.8), bei der das Gatter die getroffene Entscheidung im vorherigen Zeitschritt berücksichtigt. Dies hat den Vorteil, dass das Modell nicht mehr homogen ist; verschiedene Beobachtungs- und Übergangswahrscheinlichkeiten werden in unterschiedlichen Zeitschritten genutzt. Es existiert immer noch ein einzelnes Modell für jeden Zustand, das durch θ_j parameterisiert wird, doch es generiert verschiedene Übergangs- oder Beobachtungswahrscheinlichkeiten in Abhängigkeit von der bereits gesehenen Eingabe. Es ist möglich, dass die Eingabe kein einzelner Wert ist, sondern ein Fenster um den Zeitpunkt t, wodurch aus der Eingabe ein Vektor wird; das ermöglicht die Bearbeitung von Applikationen, bei denen Eingabe und Beobachtungssequenzen unterschiedlich lang sind.

Markovsche gemischte Expertensysteme

Selbst wenn es keine andere explizite Eingabesequenz gibt, kann ein HMM mit Eingabe genutzt werden, indem eine „Eingabe“ durch eine vorspezifizierte Funktion der vorangegangenen Beobachtungen generiert wird:

$$\boldsymbol{x}_t = \boldsymbol{f}(O_{t-\tau}, \ldots, O_{t-1}) ,$$

wodurch ein Fenster der Größe τ mit kontextbezogener Eingabe bereitgestellt wird.

FAKTORIELLES HMM

STAMMBAUM

Ein anderer HMM-Typ, der leicht visualisiert werden kann, ist das *faktorielle HMM*, bei dem es mehrere getrennte Sequenzen gibt, durch deren Wechselwirkung eine einzelne Beobachtungssequenz erzeugt wird. Ein Beispiel hierfür ist ein *Stammbaum,* der die Eltern-Kind-Beziehungen aufzeigt (Jordan 2004). Abbildung 15.6b modelliert die *Meiose*, bei der die beiden Sequenzen die Chromosomen von Vater und Mutter sind (sie sind also unabhängig), und jeder Genlocus des Nachkommen bekommt ein Allel vom Vater und das andere von der Mutter.

GEKOPPELTES HMM

Ein *gekoppeltes HMM* wie das in Abbildung 15.6c gezeigte modelliert zwei parallele, aber wechselwirkende verborgene Sequenzen. Beispielsweise können wir bei der Spracherkennung eine beobachtete akustische Sequenz von gesprochenen Wörtern und eine beobachtete visuelle Sequenz von Lippenbildern haben, von denen jede ihre verborgenen Zustände dort hat, wo beide unabhängig sind.

SCHALTENDES HMM

In einem *schaltenden HMM,* wie es in Abbildung 15.6d gezeigt ist, gibt es K parallele, unabhängige verborgene Zustandssequenzen, und die Zustandsvariable S wählt zu einem gegebenen Zeitpunkt eine davon aus, die dann die Ausgabe generiert. Das heißt, wir schalten zwischen den Zustandssequenzen hin und her, während wir voranschreiten.

KALMAN-FILTER

LINEARES DYNAMISCHES SYSTEM

In HMMs ist die Zustandsvariable diskret, obwohl die Beobachtung kontinuierlich sein kann. Dagegen sind beim sogenannten *Kalman-Filter*, einem *linearen dynamischen System*, sowohl der Zustand als auch die Beobachtungen kontinuierlich. Im Standardfall ist der Zustand zur Zeit t eine lineare Funktion des Zustands zur Zeit $t-1$ mit einem additiven, um null zentrierten Gaußschen Rauschen, und in jedem Zustand ist die Beobachtung eine andere lineare Funktion des Zustands mit einem additiven, um null zentrierten Gaußschen Rauschen. Die beiden linearen Abbildungen und die Kovarianzen der beiden Rauschquellen bilden die Parameter. Alle HMM-Varianten, die wir weiter vorn diskutiert haben, können in ähnlicher Weise verallgemeinert werden, so dass sie auch für kontinuierliche Zustände anwendbar sind.

Durch geeignetes Modifizieren des Graphenmodells können wir die Architektur an die Gegebenheiten des Prozesses anpassen, welcher die Daten generiert. Dieser Prozess der Anpassung des Modells an die Daten ist eine Modellauswahlprozedur, die angewendet wird, um den bestmögliche Kompromiss zwischen Verzerrung und Varianz zu finden. Der Nachteil ist,

dass exakte Inferenz auf derart ausgedehnten HMMs unter Umständen nicht mehr möglich ist und wir in diesem Fall Approximationen oder Sampling-Verfahren anwenden müssen (Ghahramani 2001; Jordan 2004).

15.10 Modellauswahl im HMM

Wie bei jedem Modell sollte auch die Komplexität eines HMM so abgestimmt werden, dass seine Komplexität mit der Größe und den Eigenschaften der vorliegenden Daten ausbalanciert wird. Eine Möglichkeit besteht darin, die Topologie des HMM abzustimmen. Bei einem vollständig verbundenen (ergodischen) HMM gibt es Übergänge von einem Zustand in jeden beliebigen anderen Zustand, welcher $\mathbf{A}$ zu einer vollen $N \times N$ Matrix macht. Bei einigen Anwendungen sind nur bestimmte Übergänge zulässig, und die nicht gestatteten sind durch $a_{ij} = 0$ ausgewiesen. Wenn es weniger mögliche Folgezustände gibt, $N' < N$, dann ist die Komplexität von Vorwärts-Rückwärts-Durchläufen und der Viterbi-Prozedur $\mathcal{O}(NN'T)$ statt $\mathcal{O}(N^2T)$.

Links-Rechts-HMM

Bei der Spracherkennung beispielsweise werden *Links-Rechts-HMM* verwendet, deren Zustände zeitlich geordnet vorliegen, so dass sich mit zunehmender Zeit der Zustandsindex erhöht oder gleich bleibt. Solch eine Einschränkung gestattet die Modellierung von Sequenzen, deren Eigenschaften sich über die Zeit hinweg verändern, so wie bei der Sprache, und wenn wir einen Zustand erlangen, kennen wir in etwa die ihm vorausgegangenen Zustände. Es besteht die Eigenschaft, dass wir nie in einen Zustand mit einem kleineren Index übergehen, sprich $a_{ij} = 0, \text{für } j < i$. Große Veränderungen in den Zustandsindices sind auch nicht zulässig, also $a_{ij} = 0, \text{für } j > i + \tau$. Das in Abbildung 15.7 gegebene Beispiel des Links-Rechts-HMM mit $\tau = 2$ hat die Zustandsübergangsmatrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} .$$

Ein weiterer Faktor, der die Komplexität eines HMM bestimmt, ist die Anzahl an Zuständen N. Weil die Zustände verborgen sind, ist ihre Anzahl unbekannt und sollte vorm Training bestimmt werden. Dies geschieht mit Hilfe vorher verfügbarer Informationen und kann durch Kreuzvalidierung feinabgestimmt werden, sprich, indem die Wahrscheinlichkeit von Validierungssequenzen überprüft wird.

Bei Verwendung für die Klassifikation haben wir eine Menge von HMM, von denen jedes die zu einer Klasse gehörigen Sequenzen modelliert. Um beispielsweise gesprochene Wörter zu erkennen, werden gesprochene Beispiele eines jeden Wortes genutzt, um daraus jeweils ein Modell λ_i zu trainieren. Bei einer gegebenen neuen Wortäußerung O, die es zu

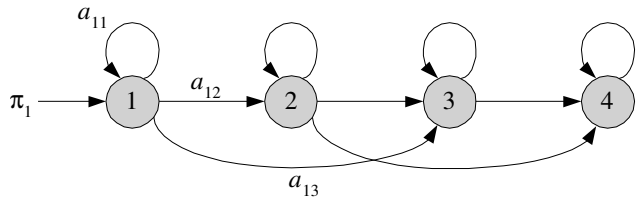


Abb. 15.7: Beispiel für ein Links-Rechts-HMM.

klassifizieren gilt, werden alle separaten Wortmodelle ausgewertet, um $P(O|\lambda_i)$ zu berechnen. Wir nutzen dann den Satz von Bayes, um die a-posteriori-Wahrscheinlichkeiten zu erhalten:

$$P(\lambda_i|O) = \frac{P(O|\lambda_i)P(\lambda_i)}{\sum_j P(O|\lambda_j)P(\lambda_j)} \tag{15.42}$$

mit $P(\lambda_i)$ als die a-priori-Wahrscheinlichkeit für das Modell λ_i, das das Word i modelliert. Die Äußerung wird dem Modell und damit dem Wort mit der höchsten a-posteriori-Wahrscheinlichkeit zugewiesen. Dies ist der Likelihood-basierte Ansatz; es existieren auch Arbeiten zu diskriminativen HMM, die direkt trainiert werden, um die a-posteriori-Wahrscheinlichkeiten zu maximieren. Wenn es mehrere Aussprachen für ein und dasselbe Wort gibt, dann werden diese als parallele Pfade im HMM für das Wort definiert.

PHONE

Im Falle einer kontinuierlichen Eingabe wie der Sprache besteht die schwierige Aufgabe, das Signal in kleine diskrete Beobachtungen zu segmentieren. Typischerweise nutzt man *Phone*, die als die primitiven Teile gewählt werden, und durch die Kombination aus ihnen entstehen längere Sequenzen (z.B. Wörter). Jedes Phon wird parallel erkannt (durch den Vektorquantisierer), dann wird das HMM verwendet, um sie seriell zu kombinieren. Sind die Sprachprimitiven einfach, dann wird das HMM komplex, und umgekehrt. Bei der Erkennung zusammenhängender Sprache, bei der die Wörter nicht eines nach dem anderen mit deutlichen Pausen zwischen ihnen geäußert werden, existiert eine HMM-Hierarchie auf mehreren Ebenen; eine Ebene kombiniert Phone, um Wörter zu erkennen, eine andere kombiniert Wörter, um Sätze zu erkennen, indem ein Sprachmodell aufgebaut wird, und so weiter.

Hybride neuronale Netzen/HMMs werden auch bei der Spracherkennung eingesetzt (Morgan und Bourlard 1995). In so einem Modell wird ein mehrlagiges Perzeptron (Kapitel 11) genutzt, um zeitlich lokale aber möglicherweise komplexe und nichtlineare Primitive zu modellieren, beispielsweise Phone, während das HMM verwendet wird, um die temporale Struktur zu lernen. Das neuronale Netz agiert als Vorbearbeitungsstufe und übersetzt die rohen Beobachtungen in einem Zeitfenster in eine Form, die leichter zu modellieren ist als die Ausgabe eines Vektorquantisierers.

Ein HMM kann durch ein Graphenmodell visualisiert werden, und die Auswertung in einem HMM ist ein Spezialfall des Belief-Propagation-Algorithmus, den wir in Kapitel 14 diskutiert haben. Der Grund, weshalb wir diesem Modell ein eigenes Kapitel widmen, ist seine große Verbreitung und seine erfolgreiche Anwendung, besonders bei der automatischen Spracherkennung. Die Standardarchitektur des HMM kann jedoch erweitert werden – beispielsweise dadurch, dass mehrere Sequenzen zugelassen werden, oder durch die Einführung von verborgenen (latenten) Variablen, was wir in Abschnitt 15.9 diskutiert haben.

In Kapitel 16 werden wir uns mit dem Bayesschen Ansatz beschäftigen, und speziell in Abschnitt 16.8 wird es um nichtparametrische Bayessche Methoden gehen, bei denen die Modellstruktur komplexer gemacht werden kann, wenn mit fortschreitender Zeit mehr Daten vorliegen. Eine Anwendung hierfür ist *das infinite HMM* (Beal, Ghahramani und Rasmussen 2002).

15.11 Anmerkungen

Das HMM ist eine ausgereifte Technologie, und es sind bereits HMM-basierte kommerzielle Spracherkennungssysteme im Umlauf (Rabiner und Juang 1993; Jelinek 1997). In Abschnitt 11.12 haben wir uns damit befasst, wie mehrlagige Perzeptronen für die Erkennung von Sequenzen trainiert werden können. Das HMM hat den Vorteil gegenüber Time Delay Neural Networks, dass kein Zeitfenster a priori definiert werden muss, und es trainiert besser als rekurrente neuronale Netze. Die HMM werden bei diversen Sequenzerkennungsaufgaben angewandt. Der Einsatz von HMM in der Bioinformatik wird bei Baldi und Brunak 1998 beschrieben, und die Anwendung bei der Verarbeitung natürlicher Sprache findet sich bei Manning und Schütze 1999. HMM werden auch bei der online Erkennung von handschriftlichen Zeichen eingesetzt, welche sich von der optischen Erkennung dahingehend unterscheidet, dass der Schreiber auf einem berührungsempfindlichen Touchpad schreibt und die Eingabe aus einer Sequenz von (x, y)-Koordinaten der Stiftspitze besteht, wenn dieser sich über das Pad bewegt; die Eingabe ist also kein statisches Bild. Bengio et al. (1995) erklären ein hybrides System für die online Erkennung, bei dem ein MLP individuelle Zeichen erkennt und ein HMM diese kombiniert, um Wörter zu erkennen. Diverse Anwendungen des HMM und mehrere Erweiterungen, beispielsweise das diskriminative HMM, werden bei Bengio 1999 besprochen. Eine neuere Übersicht über das, was HMM können und was nicht, bietet Bilmes 2006.

Ein kritischer Punkt in einem solchen Erkennungssystem ist die Entscheidung, wie viele Teilaufgaben parallel ausgeführt werden sollen und was der seriellen Verarbeitung überlassen bleibt. Bei der Spracherkennung können Phoneme durch ein paralleles System erkannt werden, was der Annahme entspricht, dass der gesamte Laut des Phonems in einem

Zeitschritt geäußert wird. Das Wort wird dann seriell erkannt, indem die Phoneme kombiniert werden. In einem alternativen System können Phoneme selbst als Sequenz von einfacheren Sprachlauten entworfen werden, wenn dasselbe Phonem viele Versionen hat, zum Beispiel in Abhängigkeit vom vorhergehenden und nachfolgenden Phonem. Die parallele Ausführung von Aufgaben ist nur bis zu einem gewissen Grad vorteilhaft; man sollte die ideale Balance zwischen paralleler und serieller Verarbeitung finden. Um in der Lage zu sein, eine beliebige Person mit einem Knopfdruck anzurufen, bräuchten wir Millionen von Tasten auf unserem Telefon; stattdessen haben wir aber zehn Tasten und drücken sie in einer bestimmten Reihenfolge, um die gewünschte Nummer zu wählen.

In Kapitel 14 haben wir Graphenmodelle behandelt, und wir wissen, dass HMMs als eine spezielle Klasse von Graphenmodellen aufgefasst werden können. Inferenz und Lernoperationen in HMMs sind daher analog zu ihren Entsprechungen in Graphenmodellen (Smyth, Heckerman und Jordan 1997). Es gibt viele Erweiterungen zu HMMs, beispielsweise faktorielle HMMs, bei denen es in jedem Zeitschritt eine Zahl an Zuständen gibt, die kollektiv die Beobachtung generieren, außerdem baumstrukturierte HMMs, bei denen es eine Hierarchie von Zuständen gibt. Der allgemeine Formalismus gestattet es auch, kontinuierliche Zustände ebenso wie diskrete zu behandeln, was auf sogenannte *lineare dynamische Systeme* führt. Bei einigen dieser Modelle ist exakte Inferenz nicht möglich und es müssen dann Approximationen und Sampling-Verfahren verwendet werden (Ghahramani 2001).

Tatsächlich kann jedes Graphenmodell zeitlich erweitert werden, indem man es in die Dimension der Zeit entfaltet und Abhängigkeiten zwischen aufeinanderfolgenden Kopien hinzufügt. Im Grunde ist ein Hidden-Markov-Modell nichts anderes als eine Folge von Clusterproblemen, wobei der Clusterindex zur Zeit t nicht nur von der Beobachtung zur Zeit t abhängt, sondern auch vom Clusterindex zur Zeit $t-1$. Der Baum-Welch-Algorithmus ist eine Expectation-Maximation-Erweiterung, mit der auch diese zeitliche Abhängigkeit berücksichtigt werden kann. In Abschnitt 6.5 haben wir uns mit der Faktorenanalyse befasst, bei der eine kleine Zahl von verborgenen Faktoren die Beobachtung generiert; ähnlich kann ein lineares dynamisches System als eine Folge solcher Faktorenmodelle aufgefasst werden, wobei die aktuellen Faktoren auch von den vorherigen abhängen.

Diese dynamische Abhängigkeit kann integriert werden, wenn dies erforderlich ist. Betrachten wir zum Beispiel noch einmal Abbildung 14.5, in der die Ursache für nasses Gras an einem bestimmten Tag modelliert wird. Wenn wir glauben, dass das Wetter von gestern einen Einfluss auf das Wetter von heute hat (und das sollten wir, denn es folgen in der Regel mehrere Tage mit bewölkten Wetter aufeinander und dann ist es mehrere Tage lang sonnig), dann können wir diese Abhängigkeit durch das in Abbildung 15.8 gezeigte dynamische Graphenmodell modellieren.

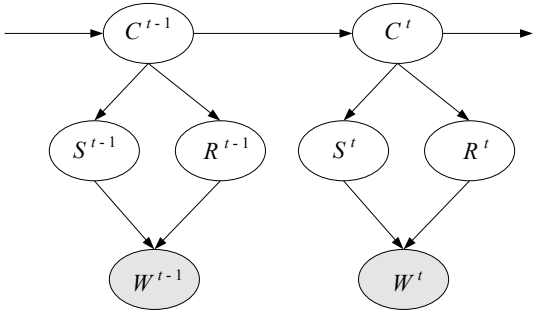


***Abb. 15.8*:** Eine dynamische Version, bei der wir eine Kette von Graphen haben, um die Abhängigkeit des Wetters an aufeinanderfolgenden Tagen darzustellen.

15.12 Übungen

1. Gegeben sei das beobachtbare Markov-Modell mit drei Zuständen S_1, S_2, S_3, die Ausgangswahrscheinlichkeiten

   $$\mathbf{\Pi} = [0{,}5, 0{,}2, 0{,}3]^T ,$$

   und die Übergangswahrscheinlichkeiten

   $$\mathbf{A} = \begin{bmatrix} 0{,}4 & 0{,}3 & 0{,}3 \\ 0{,}2 & 0{,}6 & 0{,}2 \\ 0{,}1 & 0{,}1 & 0{,}8 \end{bmatrix} .$$

   Generieren Sie 100 Sequenzen von 1.000 Zuständen.

2. Schätzen Sie Π, $\mathbf{A}$ unter Verwendung der in der vorhergehenden Übung generierten Daten und vergleichen Sie dies mit den zur Generierung der Daten verwendeten Parametern.

3. Entwerfen Sie ein Markov-Modell zweiter Ordnung. Was sind die Parameter? Wie können wir die Wahrscheinlichkeit einer gegebenen Zustandssequenz berechnen? Wie können die Parameter für den Fall eines beobachtbaren Modells gelernt werden?

   LÖSUNG: In einem Modell zweiter Ordnung hängt der aktuelle Zustand von den beiden vorherigen Zuständen ab:

   $$a_{ijk} \equiv P(q_{t+2} = S_k | q_{t+1} = S_j, q_t = S_i) .$$

   Die Anfangswahrscheinlichkeit definiert die Wahrscheinlichkeit des ersten Zustands:

   $$\pi_i \equiv P(q_1 = S_i) .$$

Außerdem brauchen wir Parameter, um die Wahrscheinlichkeit des zweiten Zustands bei gegebenem ersten Zustand zu definieren:

$$\theta_{ij} \equiv P(q_2 = S_j | q_1 = S_i) \, .$$

Für ein gegebenes beobachtbares Markov-Modell zweiter Ordnung mit den Parametern $\lambda = (\Pi, \Theta, \mathbf{A})$ ist die Wahrscheinlichkeit einer beobachteten Zustandssequenz

$$\begin{aligned} P(O = Q|\lambda) &= P(q_1) \, P(q_2|q_1) \prod_{t=3}^{T} P(q_t|q_{t-1}, q_{t-2}) \\ &= \pi_{q1} \theta_{q_2 q_1} a_{q_3 q_2 q_1} a_{q_4 q_3 q_2} \cdots a_{q_T q_{T-1} q_{T-2}} \, . \end{aligned}$$

Die Wahrscheinlichkeiten werden als Anteile geschätzt:

$$\begin{aligned} \hat{\pi}_i &= \frac{\sum_k 1(q_1^k = S_i)}{K} \, , \\ \hat{\theta}_{ij} &= \frac{\sum_k 1(q_2^k = S_j \text{ und } q_1^k = S_i)}{1(q_1^k = S_i)} \, , \\ \hat{a}_{ijk} &= \frac{\sum_k \sum_{t=3}^{T} 1(q_t^k = S_k \text{ und } q_{t-1}^k = S_j \text{ und } q_{t-2}^k = S_i)}{\sum_k \sum_{t=3}^{T} 1(q_{t-1}^k = S_j \text{ und } q_{t-2}^k = S_i)} \, . \end{aligned}$$

4. Zeigen Sie, dass ein beliebiges Markov-Modell zweiter (oder höherer) Ordnung in ein Markov-Modell erster Ordnung konvertiert werden kann.

   Lösung: In einem Modell zweiter Ordnung hängt jeder Zustand von zwei Vorgängerzuständen ab. Wir können eine neue Menge von Zuständen definieren, die dem kartesischen Produkt der originalen Zustandsmenge mit sich selbst entspricht. Ein Modell erster Ordnung, welches auf diesen neuen N^2 Zuständen definiert wird, entspricht einem Modell zweiter Ordnung, das auf den originalen N Zuständen definiert ist.

5. Einige Forscher definieren ein Markov-Modell als die Generierung einer Beobachtung während der Traversierung einer Kante, statt bei der Ankunft in einem Zustand. Ist dieses Modell auf irgendeine Art stärker als das, was wir diskutiert haben?

   Lösung: Ähnlich wie bei der vorherigen Übung können wir neue Zustände definieren, wenn die Ausgabe nicht nur vom aktuellen Zustand abhängt, sondern auch vom nächsten Zustand. Die neuen Zustände entsprechen diesem Paar und ihre Ausgabe wird von diesem (Verbund-)Zustand generiert.

6. Generieren Sie Trainings- und Validierungssequenzen mit einem HMM Ihrer Wahl. Trainieren Sie dann verschiedene HMM, indem

Sie die Zahl an verborgenen Zuständen am selben Trainingsdatensatz variieren und berechnen Sie die Likelihood-Werte bei der Validierung. Beobachten Sie, wie die Likelihood der Validierung sich mit zunehmender Zahl an Zuständen ändert.

7. Wenn wir in Gleichung 15.38 multivariate Beobachtungen haben, wie sehen dann die Gleichungen für den M-Schritt aus?

   LÖSUNG: Für d-dimensionale $\boldsymbol{O}_t \in \mathbb{R}^d$, die aus einer d-variaten Gauß-Verteilung mit

   $$p(\boldsymbol{O}_t|q_t = S_j, \lambda) \sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

   gezogen werden, lauten die Gleichungen für den M-Schritt

   $$\hat{\boldsymbol{\mu}}_j = \frac{\sum_t \gamma_t(j)\boldsymbol{O}_t}{\sum_t \gamma_t(j)} ,$$
   $$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_t \gamma_t(j)(\boldsymbol{O}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{O}_t - \hat{\boldsymbol{\mu}}_j)^T}{\sum_t \gamma_t(j)} .$$

8. Betrachten Sie noch einmal das Urnenmodell, wobei die Kugeln jetzt *ohne Ersetzung* gezogen werden sollen. Was wird sich dadurch ändern?

   LÖSUNG: Wenn wir die Kugeln ohne Ersetzung ziehen, dann ändert sich mit jeder Iteration die Anzahl der Kugeln und damit auch die Beobachtungswahrscheinlichkeiten **B**. Wir haben dann kein homogenes Modell mehr.

9. Angenommen, wir haben zu jedem Zeitpunkt zwei Beobachtungen aus zwei verschiedenen Alphabeten. Das könnten zum Beispiel die Tageswerte von zwei Wechselkursen sein. Wie können wir diese Situation in einem HMM implementieren?

   LÖSUNG: Womit wir es in einem solchen Fall zu tun haben, ist ein verborgener Zustand, der zwei verschiedene Beobachtungen generiert. Das heißt, wir haben zwei **B**, die jeweils mit einer eigenen Beobachtungssequenz trainiert wurden. Diese beiden Beobachtungen müssen dann kombiniert werden, um **A** und π zu schätzen.

10. Wie können wir zu einem inkrementellen HMM gelangen, bei dem wenn nötig neue verborgene Zustände hinzugenommen werden?

    LÖSUNG: Dies ist wieder eine Suche im Zustandsraum. Unser Ziel ist es, die Validierungs-Log-Likelihood zu maximieren, wobei wir mithilfe eines Operators einen verborgenen Zustand hinzufügen können. Dann führen wir eine Vorwärtssuche durch. Lernalgorithmen zum Lernen der Struktur für allgemeinere Graphenmodelle haben wir in Kapitel 14 diskutiert.

15.13 Literaturangaben

Baldi, P., and S. Brunak. 1998. *Bioinformatics: The Machine Learning Approach.* Cambridge, MA: The MIT Press.

Beal, M. J., Z. Ghahramani, and C. E. Rasmussen. 2002. „The Infinite Hidden Markov Model." In *Advances in Neural Information Processing Systems 14,* ed. T. G. Dietterich, S. Becker, and Z. Ghahramani, 577–585. Cambridge, MA: MIT Press.

Bengio, Y. 1999. „Markovian Models for Sequential Data." *Neural Computing Surveys* 2: 129–162.

Bengio, Y., and P. Frasconi. 1996. „Input-Output HMMs for Sequence Processing." *IEEE Transactions on Neural Networks* 7: 1231–1249.

Bengio, Y., Y. Le Cun, C. Nohl, and C. Burges. 1995. „LeRec: A NN/HMM Hybrid for On-Line Handwriting Recognition." *Neural Computation* 7: 1289–1303.

Bilmes, J. A. 2006. „What HMMs Can Do." *IEICE Transactions on Information and Systems* E89-D: 869–891.

Ghahramani, Z. 2001. „An Introduction to Hidden Markov Models and Bayesian Networks." *International Journal of Pattern Recognition and Artificial Intelligence* 15: 9–42.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition.* Cambridge, MA: The MIT Press.

Jordan, M. I. 2004. „Graphical Models." *Statistical Science* 19: 140–155.

Manning, C. D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing.* Cambridge, MA: The MIT Press.

Meila, M., and M. I. Jordan. 1996. „Learning Fine Motion by Markov Mixtures of Experts." In *Advances in Neural Information Processing Systems 8,* ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 1003–1009. Cambridge, MA: The MIT Press.

Morgan, N., and H. Bourlard. 1995. „Continuous Speech Recognition: An Introduction to the Hybrid HMM/Connectionist Approach." *IEEE Signal Processing Magazine* 12: 25–42.

Smyth, P., D. Heckerman, and M. I. Jordan. 1997. „Probabilistic Independence Networks for Hidden Markov Probability Models." *Neural Computation* 9: 227–269.

Rabiner, L. R. 1989. „A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE* 77: 257–286.

Rabiner, L. R., and B. H. Juang. 1986. „An Introduction to Hidden Markov Models.“ *IEEE Acoustics, Speech, and Signal Processing Magazine* 3: 4–16.

Rabiner, L. R., and B. H. Juang. 1993. *Fundamentals of Speech Recognition.* New York: Prentice Hall.

16 Bayessche Schätzung

Beim Bayesschen Ansatz betrachten wir Parameter als Zufallsvariablen mit einer Verteilung, die es uns erlaubt, unsere Unsicherheit bei ihrer Schätzung zu modellieren. Wir fahren dort fort, wo wir in Abschnitt 4.4 aufgehört haben, und diskutieren das Schätzen sowohl der Parameter einer Verteilung als auch der Modellparameter für die Regression, die Klassifikation, die Clusteranalyse oder die Dimensionalitätsreduktion. Wir werden auch auf die nichtparametrische Bayessche Modellbildung eingehen, bei der die Komplexität des Modells nicht fest ist, sondern von den Daten abhängt.

16.1 Einführung

Die Bayessche Schätzung, die in Abschnitt 4.4 eingeführt wurde, behandelt einen Parameter θ als eine Zufallsvariable mit einer Wahrscheinlichkeitsverteilung. Beim Maximum-Likelihood-Ansatz, den wir in Abschnitt 4.2 behandelt haben, wird ein Parameter als eine unbekannte Konstante angesehen. Wenn wir beispielsweise den Mittelwert μ schätzen wollen, ist der entsprechende Maximum-Likelihood-Schätzer das Stichprobenmittel $\bar{X}$. Wir berechnen $\bar{X}$ über unsere Trainingsmenge, setzen den Wert in unser Modell ein und verwenden dieses zum Beispiel für die Klassifikation. Wir wissen allerdings, dass der Maximum-Likelihood-Schätzer insbesondere für kleine Stichproben ein schlechter Schätzer mit großer Streuung sein kann – wenn die Trainingsmenge variiert, dann kann es passieren, dass wir verschiedene Werte von $\bar{X}$ berechnen, was zu unterschiedlichen Diskriminanzfunktionen mit unterschiedlichen Genauigkeiten bei der Generalisierung führen kann.

Bei der Bayesschen Schätzung machen wir Gebrauch von der Tatsache, dass es eine Unsicherheit beim Schätzen von θ gibt, und verwenden anstelle eines einzelnen Wertes θ_{ML} alle θ, gewichtet nach unserer geschätzten Verteilung, $p(\theta|\mathcal{X})$. Das heißt, wir mitteln über unsere Unsicherheit beim Schätzen von θ.

Bei der Schätzung von $p(\theta|\mathcal{X})$ verwenden wir die a-priori-Informationen, die wir bezüglich des Parameterwertes haben. Solche Informationen sind besonders wichtig, wenn wir nur eine kleine Stichprobe haben (und wenn die Varianz des Maximum-Likelihood-Schätzers groß ist). In so einer Situation sind wir interessiert daran zu kombinieren, was uns die

Daten mitteilen, also den aus der Stichprobe berechneten Wert und unsere a-priori-Informationen. Wie wir bereits bei der Diskussion in Abschnitt 4.4 erörtert haben, codieren wir diese Informationen unter Verwendung einer *a-priori-Wahrscheinlichkeitsverteilung.* Wenn wir etwa anhand einer Stichprobe den Mittelwert schätzen wollen, könnten wir beispielsweise die a-priori-Vermutung haben, dass er nahe bei 2, zwischen 1 und 3, liegt, und wir schreiben dann $p(\mu)$ in einer Weise, dass der größte Teil der Dichte im Intervall $[1, 3]$ liegt.

A-PRIORI-WAHRSCHEINLICHKEIT

Wir wenden den Satz von Bayes an und kombinieren die a-priori-Wahrscheinlichkeit und die Likelihood, um die Verteilung der *a-posteriori-Wahrscheinlichkeit* zu berechnen:

A-POSTERIORI-WAHRSCHEINLICHKEIT

$$p(\theta|\mathcal{X}) = \frac{p(\theta)\, p(\mathcal{X}|\theta)}{p(\mathcal{X})} \,. \tag{16.1}$$

Hierbei ist $p(\theta)$ die a-priori-Dichte. Sie ist das, was wir über die möglichen Werte wissen, die θ annehmen kann, *bevor* wir auf die Stichprobe schauen. $p(\mathcal{X}|\theta)$ ist die *Stichproben-Likelihood.* Diese sagt uns, wie wahrscheinlich unsere Stichprobe $\mathcal{X}$ ist, wenn der Parameter der Verteilung den Wert θ annimmt. Wenn die Instanzen in unserer Stichprobe zum Beispiel zwischen 5 und 10 liegen, dann ist eine solche Stichprobe wahrscheinlich, falls μ gleich 7 ist, aber weniger wahrscheinlich, falls μ gleich 3 ist und noch unwahrscheinlicher, falls μ gleich 1 ist. Das $p(\mathcal{X})$ im Nenner dient der Normierung und stellt sicher, dass die a-posteriori-Wahrscheinlichkeit $p(\theta|\mathcal{X})$ aufintegriert 1 ergibt. Diese heißt a-posteriori-Wahrscheinlichkeit, weil sie angibt, wie wahrscheinlich θ einen bestimmten Wert annimmt, nachdem die Stichprobe beobachtet wurde. Der Satz von Bayes nimmt die a-priori-Verteilung, kombiniert sie mit dem, was die Daten offenbaren und generiert daraus die a-posteriori-Verteilung. Diese a-posteriori-Verteilung verwenden wir dann später für die Inferenz.

Angenommen, wir haben eine frühere Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$, die aus einer Verteilung mit unbekanntem Parameter θ gezogen wurde. Wir können dann eine weitere Instanz x' ziehen und würden gern ihre Wahrscheinlichkeitsverteilung berechnen. Dies können wir durch ein Graphenmodell (Kapitel 14) visualisieren, wie es in Abbildung 16.1 gezeigt ist. Wir sehen hier ein *generatives Modell*, welches darstellt, wie die Daten generiert werden. Wir nehmen zuerst θ aus $p(\theta)$ und ziehen dann Stichproben aus $p(x|\theta)$, um zuerst die Trainingsinstanzen $\boldsymbol{x}^t$ und dann die neue Testinstanz x' zu ziehen.

GENERATIVES MODELL

Wir schreiben die Verbundwahrscheinlichkeit als

$$p(x', \mathcal{X}, \theta) = p(\theta)\, p(\mathcal{X}|\theta)\, p(x'|\theta) \,.$$

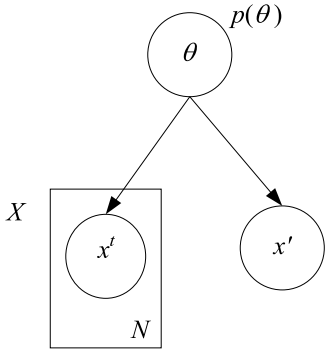


***Abb. 16.1*:** Das generative Graphenmodell (siehe Kapitel 14). Die Kanten verlaufen in Richtung der Stichprobe; wir wählen zuerst θ aus $p(\theta)$ und generieren dann Daten durch Stichproben aus $p(x|\theta)$. Die rechteckige Tafel enthält N unabhängig gezogene Instanzen, die die Trainingsmenge $\mathcal{X}$ bilden. Das neue x' wird unabhängig bei gegebenem θ gezogen. Dies ist die Annahme der unabhängigen, identischen Verteilung. Wenn θ nicht bekannt ist, sind sie abhängig. Wir inferieren θ aus den vorherigen Instanzen mithilfe des Bayesschen Satzes und wenden diese dann an, um das neue x' zu inferieren.

Wir können die Wahrscheinlichkeitsverteilung für das neue x gegeben die Stichprobe $\mathcal{X}$ schätzen:

$$p(x'|\mathcal{X}) = \frac{p(x',\mathcal{X})}{p(\mathcal{X})} = \frac{\int p(x',\mathcal{X},\theta)\,d\theta}{p(\mathcal{X})} = \frac{\int p(\theta)\,p(\mathcal{X}|\theta)\,p(x'|\theta)\,d\theta}{p(\mathcal{X})}$$
$$= \int p(x'|\theta)\,p(\theta|\mathcal{X})\,d\theta\,. \tag{16.2}$$

Bei der Berechnung von $p(\theta|\mathcal{X})$ invertiert der Bayessche Satz die Richtung der Kante und ermöglicht eine diagnostische Inferenz. Diese inferierte (a-posteriori-)Verteilung wird dann verwendet, um eine voraussagende Verteilung für neue x herzuleiten.

Wie wir sehen, ist unsere Schätzung eine gewichtete Summe (wir ersetzen $\int d\theta$ durch $\sum_\theta$, falls θ diskrete Werte annimmt) von Schätzungen, die alle möglichen Werte von θ nimmt, gewichtet danach, wie wahrscheinlich θ gegeben die Stichprobe $\mathcal{X}$ ist.

Dies ist die vollständige Bayessche Behandlung, und es kann sein, dass sie nicht möglich ist, wenn die a-posteriori-Verteilung nicht einfach integriert werden kann. Wie wir in Abschnitt 4.4 gesehen haben, verwenden wir im Falle der *Maximum-a-posteriori-Schätzung* (MAP) den Modalwert der a-posteriori-Verteilung:

MAXIMUM-A-POSTERIORI-SCHÄTZUNG

$$\theta_{MAP} = \arg\max_\theta p(\theta|\mathcal{X}) \quad \text{und} \quad p_{MAP}(x'|\mathcal{X}) = p(x'|\theta_{MAP})\,.$$

Die MAP-Schätzung entspricht der Annahme, dass die a-posteriori-Verteilung einen sehr schmalen Peak um einen einzelnen Wert, den Modalwert, hat. Wenn die a-priori-Verteilung $p(\theta)$ gleichmäßig über alle θ ist, dann liegen der Modalwert der a-posteriori-Verteilung $p(\theta|\mathcal{X})$

und der Modalwert der Likelihood $p(\mathcal{X}|\theta)$ im gleichen Punkt, und die MAP-Schätzung ist gleich der Maximum-Likelihood-Schätzung (ML):

$$\theta_{ML} = \arg\max_{\theta} p(\mathcal{X}|\theta) \quad \text{und} \quad p_{ML}(x'|\mathcal{X}) = p(x'|\theta_{ML}) \; .$$

Das bedeutet, dass das Verwenden der ML der Annahme entspricht, dass es keinen a-priori-Unterschied zwischen verschiedenen Werten von θ gibt.

Grundsätzlich hat der Bayessche Ansatz zwei Vorteile:

1. Die a-priori-Verteilung hilft uns dabei, diejenigen Werte zu ignorieren, die θ nur mit sehr geringer Wahrscheinlichkeit annimmt, und uns auf den Bereich zu konzentrieren, in dem es mit großer Wahrscheinlichkeit liegt. Selbst eine flach abfallende a-priori-Verteilung kann sehr nützlich sein.

2. Anstatt bei der Vorhersage eine einzelne θ-Schätzung zu verwenden, generieren wir einen Satz von möglichen θ-Werten (definiert durch die a-posteriori-Verteilung) und verwenden für die Vorhersage alle diese Werte, gewichtet nach ihrer Wahrscheinlichkeit.

Wenn wir die MAP-Schätzung verwenden anstatt über θ zu integrieren, dann nutzen wir den ersten der beiden Vorteile aus, aber nicht den zweiten – wenn wir die ML-Schätzung verwenden, verlieren wir beide Vorteile. Wenn wir eine nicht-informative (gleichmäßige) a-priori-Verteilung verwenden, dann nutzen wir den zweiten Vorteil, aber nicht den ersten. Tatsächlich ist es der zweite Vorteil und nicht der erste, der den Bayesschen Ansatz interessant macht. In Kapitel 17, in dem wir uns mit Kombinationen von Modellen beschäftigen, werden wir Verfahren kennenlernen, die sehr ähnlich, wenn auch nicht immer Bayessch sind.

Dieser Ansatz kann mit unterschiedlichen Typen von Verteilungen und für unterschiedliche Arten von Anwendungen benutzt werden. Der Parameter θ kann der Parameter einer Verteilung sein. Bei der Klassifikation zum Beispiel kann er das unbekannte Klassenmittel sein, für das wir einen a-priori-Wert definieren und den zugehörigen a-posteriori-Wert erhalten. Wir bekommen dann für jeden möglichen Wert des Mittels eine andere Diskriminanzfunktion, und folglich wird der Bayessche Ansatz über alle möglichen Diskriminanzfunktionen mitteln, während es beim ML-Ansatz eine einzige Schätzung für den Mittelwert gibt und daher auch nur eine Diskriminanzfunktion.

Der unbekannte Parameter kann, wie wir in Kürze sehen werden, auch der Parametersatz eines angepassten Modells sein. Beispielsweise können wir bei der linearen Regression eine a-priori-Verteilung für den Anstieg und die Regressionsparameter definieren und daraus eine a-posteriori-Verteilung berechnen, d. h. eine Verteilung für Geraden. Dann mitteln wir über die Vorhersagen aller möglichen Geraden, gewichtet nach der

Wahrscheinlichkeit, mit der sie durch ihre a-priori-Gewichte spezifiziert werden und danach, wie gut sie die gegebenen Daten anpassen.

Einer der kritischsten Aspekte der Bayesschen Schätzung ist die Auswertung des Integrals in Gleichung 16.2. In einigen Fällen können wir es berechnen, meistens jedoch nicht, und dann müssen wir es approximieren. In den nächsten Abschnitten werden wir Verfahren hierfür vorstellen, und zwar das Laplace-Verfahren, das Variationsverfahren und die Markov-Ketten-Monte-Carlo-Simulation.

Im Folgenden wollen wir uns diese und andere Anwendungen des Bayesschen Ansatzes genauer ansehen, wobei wir mit einem einfachen Fall beginnen und dann zunehmend komplexere Probleme diskutieren.

16.2 Bayessche Schätzung der Parameter diskreter Verteilungen

16.2.1 $K > 2$-Zustände: Dirichlet-Verteilung

Nehmen wir an, dass jede Instanz eine multinomiale Variable ist, die einen von K verschiedenen Zuständen annehmen kann (Abschnitt 4.2.2). Es sei $x_i^t = 1$, wenn Instanz t im Zustand i ist und $x_j^t = 0, \forall j \neq i$. Die Parameter sind die Wahrscheinlichkeiten der Zustände, $\boldsymbol{q} = [q_1, q_2, \ldots, q_K]^T$ mit $q_i, i = 1, \ldots, K$, für die $q_i \geq 0, \forall i$ und $\sum_i q_i = 1$ erfüllt ist.

Beispielsweise kann x^t neuen Dokumenten entsprechen und die Dokumenten können K verschiedene Nachrichtenrubriken sein: Sport, Politik, Kultur usw. Die Wahrscheinlichkeiten q_i entsprechen dann den Anteilen der jeweiligen Rubriken, und a-priori-Werte für sie erlauben es uns, unsere a-priori-Annahmen bezüglich der Anteile zu codieren. Beispielsweise könnten wir erwarten, dass es mehr Nachrichten in der Rubrik Sport gibt als in der Rubrik Kultur.

Die Stichproben-Likelihood ist

$$p(\mathcal{X}|\boldsymbol{q}) = \prod_{t=1}^{N} \prod_{i=1}^{K} q_i^{x_i^t} .$$

Die a-priori-Verteilung von $\boldsymbol{q}$ ist die *Dirichlet-Verteilung*

DIRICHLET-VERTEILUNG

$$\text{Dirichlet}(\boldsymbol{q}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_K)} \prod_{i=1}^{K} q_i^{\alpha_i - 1}$$

mit $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_K]^T$ und $\alpha_0 = \sum_i \alpha_i$. Die Parameter α_i der a-priori-Verteilung werden als *Hyperparameter* bezeichnet. $\Gamma(x)$ ist die

GAMMAFUNKTION *Gammafunktion,* die durch

$$\Gamma(x) \equiv \int_0^\infty u^{x-1} e^{-u}\, du$$

definiert ist.

Wenn die a-priori-Werte und die Likelihood gegeben sind, können wir die a-posteriori-Verteilung ableiten:

$$\begin{aligned} p(\boldsymbol{q}|\mathcal{X}) &\propto p(\mathcal{X}|\boldsymbol{q})\, p(\boldsymbol{q}|\boldsymbol{\alpha}) \\ &\propto \prod_i q_i^{\alpha_i + N_i - 1} \,. \end{aligned} \tag{16.3}$$

Dabei ist $N_i = \sum_{t=1}^N x_i^t$. Wir stellen fest, dass die a-posteriori-Verteilung die gleiche Form hat wie die a-priori-Verteilung. Eine solche a-priori-Verteilung nennen wir *konjugierte a-priori-Verteilung* oder kurz *konjugierter Prior.* Sowohl die a-priori-Verteilung als auch die Likelihood haben die Form eines Produktes von Potenzen der q_i, und wir kombinieren sie, um die a-posteriori-Verteilung zu bilden:

KONJUGIERTER PRIOR

$$\begin{aligned} p(\boldsymbol{q}|\mathcal{X}) &= \frac{\Gamma(\alpha_0 + N)}{\Gamma(\alpha_1 + N_1) \cdots \Gamma(\alpha_K + N_K)} \prod_{i=1}^K q_i^{\alpha_i + N_i - 1} \\ &= \text{Dirichlet}(\boldsymbol{q}|\boldsymbol{\alpha} + \boldsymbol{n}) \,. \end{aligned} \tag{16.4}$$

Hierbei ist $\boldsymbol{n} = [N_1, \ldots, N_K]^T$ und $\sum_i N_i = N$.

Ein Blick auf Gleichung 16.3 bringt uns auf eine Interpretation der Hyperparameter α_i (Bishop 2006). Ähnlich wie die n_i Zähler für das Auftreten der Zustände i in einer Stichprobe der Länge N sind, können wir die α_i als Zähler für das Auftreten der Zustände i in einer imaginären Stichprobe von α_0 Instanzen ansehen. Bei der Definition der a-priori-Verteilung sind wir subjektiv, indem wir Folgendes sagen: In einer Stichprobe des Umfangs α_0 erwarte ich, dass α_i Instanzen im Zustand i sind. Dabei bedeuten größere α_0, dass wir eine höhere Konfidenz (also eine Verteilung mit ausgeprägterem Peak) in unseren subjektiven Anteilen haben: Zu sagen „Ich erwarte, dass 60 von 100 Instanzen zum Zustand 1 gehören", hat eine höhere Konfidenz als „Ich erwarte, dass 6 von 10 zu 1 gehören". Die a-posteriori-Verteilung ist dann eine andere Dirichlet-Verteilung, die über die Zähler für das Auftreten von Zuständen summiert, imaginären wie tatsächlichen, die durch die a-priori-Verteilung bzw. die Likelihood gegeben sind.

Die Konjugiertheit, d. h. die Tatsache, dass a-posteriori-Verteilung und a-priori-Verteilung die gleiche Form haben, hat eine nützliche Implikation. In einer sequenziellen Versuchsanordnung, bei der wir eine Sequenz von Instanzen erhalten, akkumuliert die aktuelle a-posteriori-Verteilung Informationen von allen zurückliegenden Instanzen und wird die a-priori-Verteilungen für die nächsten Instanz.

16.2.2 $K = 2$-Zustände: Betaverteilung

Wenn die Variable binär verteilt ist, also $x^t \in \{0, 1\}$, dann hat die multinomiale Stichprobe eine Bernoulli-Verteilung

$$p(\mathcal{X}|q) = \prod_t q^{x^t}(1-q)^{1-x^t}$$

und die Dirichlet-a-priori-Verteilung reduziert sich auf die *Betaverteilung* BETAVERTEILUNG

$$\text{Beta}(q|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} q^{\alpha-1}(1-q)^{\beta-1} .$$

Beispielsweise kann x^t gleich 0 oder 1 sein, je nachdem, ob eine E-Mail, die in einer zufälligen Stichprobe der Länge N den Index t hat, eine legitime Mail oder Spam ist. Die Definition einer a-priori-Verteilung von q erlaubt uns dann, eine a-priori-Annahme über die Spam-Wahrscheinlichkeit zu formulieren: „Ich erwarte, dass im Mittel $\alpha/(\alpha+\beta)$ von meinen E-Mails Spam sind."

Die Betaverteilung ist eine konjugierte a-priori-Verteilung, und für die a-posteriori-Verteilung erhalten wir

$$p(q|A, N, \alpha, \beta) \propto q^{A+\alpha-1}(1-p)^{N-A+\beta-1}$$

mit $A = \sum_t x^t$. Wir stellen wieder fest, dass die Vorkommnisse in imaginären und tatsächlichen Stichproben kombiniert werden. Man beachte, dass wir für $\alpha = \beta = 1$ eine gleichmäßige a-priori-Verteilung haben und die a-posteriori-Verteilung die gleiche Form wie die Likelihood hat. Wenn die beiden Zähler wachsen, egal ob α und β für die a-priori-Verteilung oder $\alpha + A$ und $\beta + N - A$ für die a-posteriori-Verteilung, und auch die Differenz zwischen ihnen wächst, erhalten wir eine Verteilung mit ausgeprägterem Peak und kleinerer Varianz (siehe Abbildung 16.2). Mit größer werdender Datenmenge (imaginär oder tatsächlich) wird die Varianz kleiner.

16.3 Bayessche Schätzung der Parameter einer Gauß-Verteilung

16.3.1 Univariater Fall: Unbekannter Mittelwert, bekannte Varianz

Wir betrachten nun den Fall, dass die Instanzen gaußverteilt sind. Beginnen wir mit dem univariaten Fall, $p(x) \sim \mathcal{N}(\mu, \sigma^2)$, in dem die Parameter

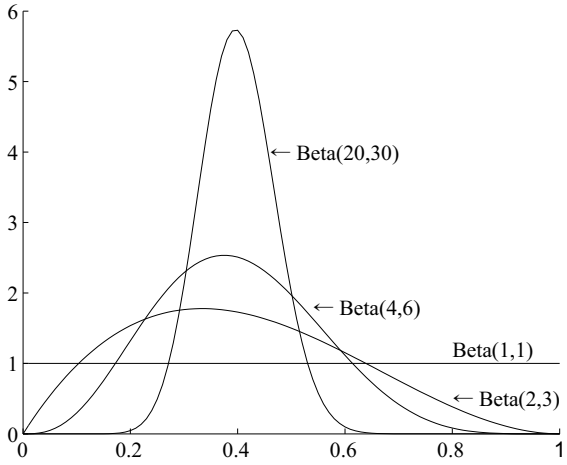


***Abb. 16.2*:** Betaverteilungen für verschiedene Werte der Parameter α und β.

μ und σ^2 sind (diesen Fall haben wir in Abschnitt 4.4 bereits kurz diskutiert). Die Stichproben-Likelihood ist

$$p(\mathcal{X}|\mu,\sigma^2) = \prod_t \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x^t-\mu)^2}{2\sigma^2}\right]. \tag{16.5}$$

Die konjugierte a-priori-Verteilung für μ ist eine Gauß-Verteilung, $p(\mu) \sim \mathcal{N}(\mu_0^2, \sigma_0^2)$, und wir schreiben die a-posteriori-Verteilung als

$$\begin{aligned} p(\mu|\mathcal{X}) &\propto p(\mu)\, p(\mathcal{X}|\mu) \\ &\sim \mathcal{N}(\mu_N, \sigma_N^2) \end{aligned}$$

mit

$$\mu_N = \frac{\sigma^2}{N\sigma_0^2+\sigma^2}\,\mu_0 + \frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\,m\,, \tag{16.6}$$

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}\,. \tag{16.7}$$

wobei $m = \sum_t x^t/N$ das Stichprobenmittel ist. Wir sehen, dass der Mittelwert μ_N der a-posteriori-Dichte (bei der es sich um die MAP-Schätzung handelt) ein gewichtetes Mittel aus dem a-priori-Mittelwert μ_0 und dem Stichprobenmittel m ist, wobei die Gewichte umgekehrt proportional zu den Varianzen sind (Abbildung 16.3 zeigt ein Beispiel). Man beachte, dass μ_N stets zwischen μ_0 und m liegt, da beide Koeffizienten zwischen 0 und 1 liegen und sich zu 1 addieren. Wenn der Umfang N der Stichprobe oder die Varianz der a-priori-Verteilung groß sind, liegt der Mittelwert

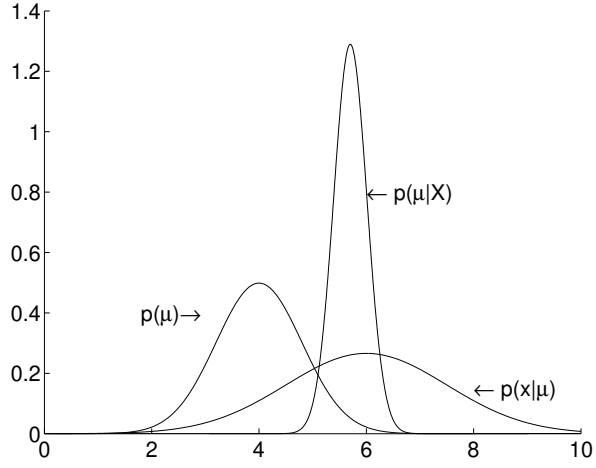


***Abb. 16.3*:** Gezeichnet sind zwanzig Datenpunkte, die aus $p(x) \sim \mathcal{N}(6, 1{,}5^2)$ gezogen wurden. Die a-priori-Verteilung ist $p(\mu) \sim \mathcal{N}(4, 0{,}8^2)$ und die a-posteriori-Verteilung ist dann $p(\mu|\mathcal{X}) \sim \mathcal{N}(5{,}7, 0{,}3^2)$.

der a-posteriori-Verteilung nahe m, d. h., er beruht vor allem auf der von der Stichprobe gelieferten Information. Wenn σ_0^2 klein ist, d. h., wenn wir wenig Unsicherheit bezüglich des korrekten Wertes von μ haben oder wenn die Stichprobe klein ist, hat unsere a-priori-Vermutung μ_0 einen stärkeren Effekt.

σ_N wird kleiner, wenn einer der Werte σ_0 oder σ kleiner wird oder wenn N größer wird. Beachten Sie außerdem, dass σ_N kleiner als σ_0 und auch als $\sigma/\sqrt{N}$ ist, d. h., die a-posteriori-Varianz ist kleiner als die a-priori-Varianz und auch kleiner als die Varianz von m. Wenn man beide berücksichtigt, wird man eine bessere a-posteriori-Schätzung erhalten als allein mit dem a-priori-Wert oder dem Stichprobenwert.

Wenn σ^2 bekannt ist, können wir für ein neues x über diese a-posteriori-Verteilung integrieren, um eine Vorhersage zu machen:

$$\begin{aligned} p(x|\mathcal{X}) &= \int p(x|\mu)\, p(\mu|\mathcal{X})\, d\mu \\ &\propto \mathcal{N}(\mu_N, \sigma_N^2 + \sigma^2)\,. \end{aligned} \tag{16.8}$$

Wie wir sehen, ist x noch immer Gaußsch, und zwar zentriert um den a-posteriori-Mittelwert, während die Varianz nun die Unsicherheit aufgrund der Schätzung des Mittelwert und der neuen Stichprobeninstanz x umfasst. Wir schreiben $x = \boldsymbol{\mu} + x'$ mit $x' \sim \mathcal{N}(0, \sigma^2)$ und erhalten dann $E[x] = E[\mu] + E[x'] = \mu_N$ und $\mathrm{Var}(x) = \mathrm{Var}(\mu) + \mathrm{Var}(x') = \sigma_N^2 + \sigma^2$, wobei Letzteres aus der Tatsache folgt, dass das neue x' unabhängig gezogen wird.

Nachdem wir eine Verteilung für $p(x|\mathcal{X})$ haben, können wir diese für verschiedene Zwecke verwenden. Bei der Klassifikation beispielsweise entspricht dieser Ansatz der Annahme Gaußscher Klassen, wobei die Mittelwerte eine Gaußsche a-priori-Verteilung haben und unter Verwendung von $\mathcal{X}_i$, der der Klasse C_i zugeordneten Teilmenge von $\mathcal{X}$, trainiert sind. Dann entspricht $p(x|\mathcal{X}_i)$, berechnet wie oben, $p(x|C_i)$, was wir mit der a-priori-Verteilung $P(C_i)$ kombinieren, um die a-posteriori-Verteilung und somit eine Diskriminanzfunktion zu erhalten.

16.3.2 Univariater Fall: Unbekannter Mittelwert, unbekannte Varianz

Wenn wir σ^2 nicht kennen, müssen wir auch diesen Parameter schätzen.

PRÄZISION Im Falle der Varianz arbeiten wir mit der *Präzision,* dem Kehrwert der Varianz, $\lambda \equiv 1/\sigma^2$. Damit kann die Likelihood wie folgt geschrieben werden:

$$\begin{aligned} p(\mathcal{X}|\lambda) &= \prod_t \frac{\lambda^{1/2}}{\sqrt{2\pi}} \exp\Big[-\frac{\lambda}{2}(x^t-\mu)^2\Big] \\ &= \lambda^{N/2}(2\pi)^{-N/2} \exp\left[-\frac{\lambda}{2}\sum_t (x^t-\mu)^2\right]. \end{aligned} \tag{16.9}$$

GAMMAVERTEILUNG Der konjugierte Prior für die Präzision ist die *Gammaverteilung:*

$$p(\lambda) \sim \text{gamma}(a_0, b_0) = \frac{1}{\Gamma(a_0)}\, b_0^{a_0} \lambda^{a_0-1} \exp(-b_0\lambda)\,.$$

Dabei definieren wir $a_0 \equiv v_0/2$ und $b_0 \equiv (v_0/2)s_0^2$, so dass s_0^2 unsere a-priori-Schätzung für die Varianz ist und v_0 unsere Konfidenz in diesen Prior – man kann diese interpretieren als die Größe der imaginären Stichprobe, von der wir glauben, dass s_0^2 aus ihr geschätzt wurde.

Der Posterior ist dann ebenfalls eine Gammaverteilung:

$$\begin{aligned} p(\lambda|\mathcal{X}) &\propto p(\mathcal{X}|\lambda)\, p(\lambda) \\ &\sim \text{gamma}(a_N, b_N) \end{aligned}$$

mit

$$\begin{aligned} a_N &= a_0 + N/2 = \frac{v_0+N}{2}\,, \\ b_N &= b_0 + \frac{N}{2}s^2 = \frac{v_0}{2}s_0^2 + \frac{N}{2}s^2\,. \end{aligned} \tag{16.10}$$

Hierbei ist $s^2 = \sum_t (x^t-\mu)^2/N$ die Stichprobenvarianz. Wieder stellen wir fest, dass a-posteriori-Schätzungen gewichtete Summen von Prior und Stichprobenstatistik sind.

Um eine Vorhersage für neue x machen zu können, wenn sowohl μ als auch σ^2 unbekannt sind, brauchen wir den Verbund-Posterior, den wir in der Form

$$p(\mu, \lambda) = p(\mu|\lambda)\, p(\lambda)$$

schreiben, wobei $p(\lambda) \sim \text{Gamma}(a_0, b_0)$ und $p(\mu|\lambda) \sim \mathcal{N}(\mu_0, 1/(\kappa_0\lambda))$. Hier kann man sich κ_0 wieder als die Größe der imaginären Stichprobe vorstellen, und als solche definiert sie unsere Konfidenz in den Prior. Der konjugierte Prior für die Verbundwahrscheinlichkeit ist in diesem Fall die sogenannte *Normal-Gamma-Verteilung*

Normal-Gamma-Verteilung

$$\begin{aligned} p(\mu, \lambda) &\sim \text{Normal-Gamma}(\mu_0, \kappa_0, a_0, b_0) \\ &= \mathcal{N}(\mu, 1/(\kappa_0\lambda)) \cdot \text{Gamma}(a_0, b_0)\,. \end{aligned}$$

Der Posterior ist

$$p(\mu, \lambda|\mathcal{X}) \sim \text{Normal-Gamma}(\mu_N, \kappa_N, a_N, b_N) \tag{16.11}$$

mit

$$\begin{aligned} \kappa_N &= \kappa_0 + N\,, \\ \mu_N &= \frac{\kappa_0\mu_0 + Nm}{\kappa_N}\,, \\ a_N &= a_0 + N/2\,, \\ b_N &= b_0 + \frac{N}{2}s^2 + \frac{\kappa_0 N}{2\kappa_N}(m - \mu_0)^2\,. \end{aligned} \tag{16.12}$$

Um eine Vorhersage für neue x zu machen, integrieren wir über den Posterior:

$$p(x|\mathcal{X}) = \iint p(x|\mu, \lambda)\, p(\mu, \lambda|\mathcal{X})\, d\mu\, d\lambda \tag{16.13}$$

$$\sim t_{2a_N}\left(\mu_N, \frac{b_N(\kappa_N + 1)}{a_N\kappa_N}\right). \tag{16.14}$$

Das heißt, wir erhalten eine (nicht standardisierte) t-Verteilung, die die gegebenen Werte für Mittelwert und Varianz mit $2a_N$ Freiheitsgraden hat. In Gleichung 16.8 haben wir eine Gauß-Verteilung; dort ist der Mittelwert derselbe, aber da σ^2 unbekannt ist, fügt seine Schätzung eine Unsicherheit hinzu, weshalb wir eine t-Verteilung mit breiteren Flanken erhalten. Manchmal wählen wir eine äquivalente Vorgehensweise, bei der nicht die Präzision λ, sondern σ^2 modelliert wird. Dabei können wir die inverse Gammaverteilung oder die inverse Chi-Quadrat-Verteilung verwenden, siehe Murphy 2007.

16.3.3 Multivariater Fall: Unbekannter Mittelwert, unbekannte Kovarianz

Wenn wir multivariate $\boldsymbol{x} \in \mathbb{R}^d$ haben, gehen wir ganz ähnlich vor, allerdings mit der Verallgemeinerung, dass wir die multivariaten Versionen der Verteilungen benutzen müssen (Murphy 2012). Wir haben

$$p(\boldsymbol{x}) \sim \mathcal{N}_d(\boldsymbol{\mu}, \boldsymbol{\Lambda}) \,,$$

Präzisionsmatrix

wobei $\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1}$ die *Präzisionsmatrix* ist. Für den Mittelwert verwenden wir einen Gaußschen Prior (bedingt unter $\boldsymbol{\Lambda}$):

$$p(\boldsymbol{\mu}|\boldsymbol{\Lambda}) \sim \mathcal{N}_d(\boldsymbol{\mu}_0, (1/\kappa_0)\boldsymbol{\Lambda})$$

Wishart-Verteilung

und für die Präzisionsmatrix die multivariate Version der Gammaverteilung, die auch *Wishart-Verteilung* genannt wird,

$$p(\boldsymbol{\Lambda}) \sim \text{Wishart}(v_0, \mathbf{V}_0) \,.$$

Dabei entspricht v_0 bzw. κ_0 dem Grad unseres Vertrauens in den Prior.

Normal-Wishart-Verteilung

Der konjugierte Verbund-Prior ist die *Normal-Wishart-Verteilung*

$$\begin{aligned} p(\mu, \boldsymbol{\Lambda}) &= p(\boldsymbol{\mu}|\boldsymbol{\Lambda})\, p(\boldsymbol{\Lambda}) \\ &\sim \text{Normal-Wishart}(\boldsymbol{\mu}_0, \kappa_0, v_0, \mathbf{V}_0) \end{aligned} \tag{16.15}$$

und der Posterior ist

$$p(\mu, \boldsymbol{\Lambda}|\mathcal{X}) \sim \text{Normal-Wishart}(\boldsymbol{\mu}_N, \kappa_N, v_N, \mathbf{V}_N)$$

mit

$$\begin{aligned} \kappa_N &= \kappa_0 + N \,, \\ \boldsymbol{\mu}_N &= \frac{\kappa_0 \boldsymbol{\mu}_0 + N\boldsymbol{m}}{\kappa_N} \,, \\ v_N &= v_0 + N \,, \\ \mathbf{V}_N &= \left(\mathbf{V}_0^{-1} + \mathbf{C} + \frac{\kappa_0 N}{\kappa_N} (\boldsymbol{m} - \boldsymbol{\mu}_0)(\boldsymbol{m} - \boldsymbol{\mu}_0)^T \right)^{-1} . \end{aligned} \tag{16.16}$$

$\mathbf{C} = \sum_t (\boldsymbol{x}^t - \boldsymbol{m})(\boldsymbol{x}^t - \boldsymbol{m})^T$ ist die Streuungsmatrix.

Um eine Vorhersage für neue x zu machen, integrieren wir über den Verbund-Posterior:

$$p(\boldsymbol{x}|\mathcal{X}) = \iint p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda})\, p(\boldsymbol{x}, \boldsymbol{\Lambda}|\mathcal{X})\, d\boldsymbol{\mu}\, d\boldsymbol{\Lambda} \tag{16.17}$$

$$\sim t_{v_N - d + 1}\left(\boldsymbol{\mu}_N, \frac{\kappa_N + 1}{\kappa_N (v_N - d + 1)} (\mathbf{V}_N)^{-1} \right) . \tag{16.18}$$

Das heißt, wir erhalten eine (nicht standardisierte) multivariate t-Verteilung, die diesen Mittelwert und diese Varianz hat, mit $v_N - d + 1$ Freiheitsgraden.

16.4 Bayessche Schätzung der Parameter einer Funktion

Wir diskutieren nun den Fall, dass wir für die Regression oder Klassifikation die Parameter schätzen, also keine Verteilung, sondern eine gewisse Funktion der Eingabe. Wieder ist unser Ansatz der, diese Parameter als Zufallsvariablen mit einer a-priori-Verteilung zu betrachten und mithilfe des Bayesschen Satzes eine a-posteriori-Verteilung zu berechnen. Wir können dann entweder das vollständige Integral auswerten, es approximieren oder die MAP-Schätzung verwenden.

16.4.1 Regression

Betrachten wir den Fall eines linearen Regressionsmodells:

$$r = \boldsymbol{w}^T\boldsymbol{x} + \epsilon \text{ mit } \epsilon \sim \mathcal{N}(0, 1/\beta) \,. \tag{16.19}$$

Dabei ist β die Präzision des additiven Rauschens (wir nehmen an, dass eine der d Eingaben immer $+1$ ist).

Die Parameter sind die Gewichte $\boldsymbol{w}$ und wir haben eine Stichprobe $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$ mit $\boldsymbol{x} \in \mathbb{R}^d$ und $r^t \in \mathbb{R}$. Wir können das Problem mithilfe einer Matrix von Eingaben und einem Vektor von gewünschten Ausgaben formulieren, also $\mathcal{X} = [\mathbf{X}, \boldsymbol{r}]$, wobei $\mathbf{X}$ eine $N \times d$-Matrix und $\boldsymbol{r}$ eine $N \times 1$-Matrix ist. Gemäß Gleichung 16.19 haben wir

$$p(r^t|\boldsymbol{x}^t, \boldsymbol{w}, \beta) \sim \mathcal{N}(\boldsymbol{w}^T\boldsymbol{x}, 1/\beta) \,.$$

Gemäß Abschnitt 4.6 ist die Log-Likelihood gegeben durch

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}|\mathcal{X}) \equiv \log p(\mathcal{X}|\boldsymbol{w}) &= \log p(\boldsymbol{r}, \mathbf{X}|\boldsymbol{w}) \\ &= \log p(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}) + \log p(\mathbf{X}) \,,\end{aligned}$$

wobei der erste Term eine Konstante ist, d. h. unabhängig von den Parametern. Den ersten Term entwickeln wir in

$$\begin{aligned}\log p(\boldsymbol{r}|\mathbf{X}, \boldsymbol{w}, \beta) &= \log \prod_t p(r^t|\boldsymbol{x}^t, \boldsymbol{w}, \beta) \qquad (16.20)\\ &= -N\log(\sqrt{2\pi}) + N\log\sqrt{\beta} - \frac{\beta}{2}\sum_t (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 \,.\end{aligned}$$

Für den Fall der ML-Schätzung stellen wir fest, dass das $\boldsymbol{w}$, welches diesen Ausdruck maximiert, oder, äquivalent dazu, welches den letzten Term minimiert, die Summe der Fehlerquadrate ist. Dies können wir schreiben als

$$\begin{aligned}E &= \sum_{t=1}^N (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 = (\boldsymbol{r} - \mathbf{X}\boldsymbol{w})^T(\boldsymbol{r} - \mathbf{X}\boldsymbol{w}) \\ &= \boldsymbol{r}^T\boldsymbol{r} - 2\boldsymbol{w}^T\mathbf{X}^T\boldsymbol{r} + \boldsymbol{w}^T\mathbf{X}^T\mathbf{X}\boldsymbol{w} \,.\end{aligned}$$

Wir bilden die Ableitung nach $\boldsymbol{w}$ und setzen diese gleich 0:

$$-2\mathbf{X}^T\boldsymbol{r} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{w} = 0 \Rightarrow \mathbf{X}^T\mathbf{X}\boldsymbol{w} = \mathbf{X}^T\boldsymbol{r} \,.$$

Daraus erhalten wir den Maximum-Likelihood-Schätzer (wir haben diesen bereits in Abschnitt 5.8 hergeleitet):

$$\boldsymbol{w}_{\mathrm{ML}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{r} \,. \tag{16.21}$$

Nachdem wir die Parameter berechnet haben, können wir die Vorhersage in Angriff nehmen. Ist $\boldsymbol{x}'$ die neue Eingabe, dann berechnet sich die Antwort zu

$$r' = \boldsymbol{w}_{\mathrm{ML}}^T\boldsymbol{x}' \,. \tag{16.22}$$

Im allgemeinen Fall eines beliebigen Modells $g(\boldsymbol{x}|\boldsymbol{w})$, etwa für ein mehrlagiges Perzeptron, bei dem $\boldsymbol{w}$ sämtliche Gewichte sind, minimieren wir, zum Beispiel mittels Gradientenabstieg:

$$E(\mathcal{X}|\boldsymbol{w}) = \left[r^t - g(x^t|\boldsymbol{w})\right]^2 \,.$$

Das $\boldsymbol{w}_{\mathrm{LSQ}}$, welches den Ausdruck minimiert, wird *Kleinste-Quadrate-Schätzung* oder *Least-Square-Schätzung* genannt. Für neue x' wird dann die Vorhersage berechnet:

$$r' = g(\boldsymbol{x}'|\boldsymbol{w}_{\mathrm{LSQ}}) \,.$$

Im Fall des Bayesschen Ansatzes definieren wir einen Gaußschen Prior für die Parameter:

$$p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, (1/\alpha)\mathbf{I}) \,.$$

Dies ist ein konjugierter Prior, und für den Posterior erhalten wir

$$p(\boldsymbol{w}|\mathcal{X}, \boldsymbol{r}) \sim \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$$

mit

$$\begin{aligned} \boldsymbol{\mu}_N &= \beta\boldsymbol{\Sigma}_N\mathbf{X}^T\boldsymbol{r} \,, \\ \boldsymbol{\Sigma}_N &= (\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1} \,. \end{aligned} \tag{16.23}$$

Um die Ausgabe für neue $\boldsymbol{x}'$ zu berechnen, integrieren wir über den vollständigen Posterior:

$$r' = \int (\boldsymbol{w}^T\boldsymbol{x}')\, p(\boldsymbol{w}|\mathcal{X}, \boldsymbol{r})\, d\boldsymbol{w} \,.$$

Das Graphenmodell hierfür ist in Abbildung 14.7 gezeigt.

Wenn wir eine Punktschätzung verwenden wollen, dann ist der MAP-Schätzer

$$\boldsymbol{w}_{\text{MAP}} = \boldsymbol{\mu}_N = \beta(\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{r}\,, \tag{16.24}$$

und bei der Berechnung der Ausgabe für die Eingabe $\boldsymbol{x}'$ ersetzen wir die Dichte durch einen einzelnen Punkt, und zwar durch den Mittelwert:

$$r' = \boldsymbol{w}_{\text{MAP}}^T\boldsymbol{x}'\,.$$

Wir können auch die Varianz unserer Schätzung berechnen:

$$\text{Var}(r') = 1/\beta + (\boldsymbol{x}')\boldsymbol{\Sigma}_N\boldsymbol{x}'\,. \tag{16.25}$$

Vergleichen wir Gleichung 16.24 mit der ML-Schätzung gemäß Gleichung 16.21, dann sehen wir, dass wir Erstere als eine Regularisierung auffassen können – wir addieren eine Konstante α zur Diagonale, damit die Matrix, die invertiert werden soll, besser konditioniert ist.

Der Prior, $p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, (1/\alpha)\mathbf{I})$, sagt aus, dass wir die Parameter nahe bei 0 erwarten und dass die Streuung umgekehrt proportional zu α ist. Für $\alpha \to 0$ haben wir einen flachen Prior und die MAP-Schätzung konvergiert gegen die ML-Schätzung.

In Abbildung 16.4 sehen wir, dass wir durch Erhöhung von α die Parameter näher an 0 drücken und dass sich die a-posteriori-Verteilung enger um den Ursprung zusammenzieht. Wenn wir β verkleinern, dann nehmen wir ein Rauschen mit größerer Varianz an, und der Posterior hat ebenfalls eine größere Varianz.

Wenn wir den Logarithmus des Posteriors bilden, dann erhalten wir

$$\begin{aligned}\log p(\boldsymbol{w}|\mathbf{X}, \boldsymbol{r}) &\propto \log p(\boldsymbol{r}|\boldsymbol{w}, \mathbf{X}) + \log p(\boldsymbol{w}) \\ &= -\frac{\beta}{2}\sum_t (r^t - \boldsymbol{w}^T\boldsymbol{x}^t)^2 - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + c\,.\end{aligned}$$

Das maximieren wir, um die MAP-Schätzung zu bekommen. Im allgemeinen Fall können wir für unser gegebenes Modell $g(\boldsymbol{x}|\boldsymbol{w})$ eine erweiterte Fehlerfunktion

$$E_{\text{Ridge}}(\boldsymbol{w}|\mathcal{X}) = \sum_t \left[r^t - g(\boldsymbol{x}^t|\boldsymbol{w})\right]^2 + \lambda\sum_i w_i^2$$

mit $\lambda \equiv \alpha/\beta$ aufschreiben. Diese Methode wird in der Statistik *Ridge-Regression* genannt. In Abschnitt 4.8 haben wir sie unter der Bezeichnung *Regularisierung* kennengelernt, und in Abschnitt 11.9 ist sie uns als *Gewichtsabbau* begegnet. Der erste Term ist der negative Logarithmus der Likelihood und der zweite Term bestraft w_i, die von 0 entfernt sind (wie vorgegeben durch das α des Priors).

RIDGE-REGRESSION

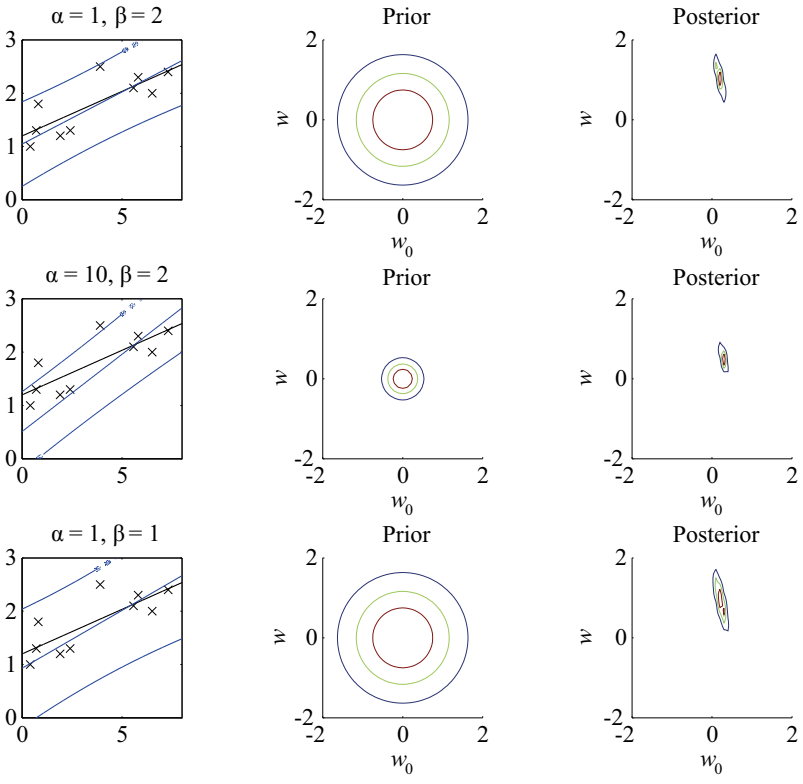


Abb. 16.4: Bayessche lineare Regression für verschiedene Werte von α und β. *Links:* Kreuze stehen für Datenpunkte und die durchgezogene Linie für die ML-Lösung. Eingezeichnet ist außerdem die MAP-Lösung mit Fehlerbalken von einer Standardabweichung (Strichlinien). *Mitte:* a-priori-Dichte mit Mittelwert 0 und Varianz $1/\alpha$. *Rechts:* a-posteriori-Dichte, deren Mittelwert die MAP-Lösung ist. Wir sehen, dass mit wachsendem α die Varianz des Priors schrumpft und die Linie sich näher zur flachen 0-Linie verschiebt. Wenn β abnimmt, wird ein stärkeres Rauschen angenommen, und die a-posteriori-Dichte hat eine größere Varianz.

Obwohl dieser Ansatz die Summe $\sum_i w_i^2$ reduziert, zwingt er nicht die einzelnen w_i gegen 0. Das bedeutet, dass er für die Merkmalsselektion nicht verwendet werden kann, bei der es darum geht zu erkennen, welche x_i redundant sind. Hierfür kann ein *Laplace-Prior* verwendet werden, der die L_1-Norm anstatt die L_2-Norm verwendet (Figueiredo 2003):

LAPLACE-PRIOR

$$p(\boldsymbol{w}|\alpha) = \prod_i \frac{\alpha}{2} \exp(-\alpha|w_i|) = \left(\frac{\alpha}{2}\right)^d \exp\left(-\alpha \sum_i |w_i|\right) .$$

Die a-posteriori-Wahrscheinlichkeit ist dann nicht mehr Gaußsch und die MAP-Schätzung wird durch Minimierung von

$$E_{\text{Lasso}}(\boldsymbol{w}|\mathcal{X}) = \sum_t (r^t - \boldsymbol{w}^T \boldsymbol{x}^t)^2 + 2\sigma^2 \alpha \sum_i |w_i|$$

ermittelt. Dabei ist σ^2 die Varianz des Rauschens (wofür wir unsere Schätzung einsetzen). Diese Methode wird *Lasso* genannt (Abk. für **l**east **a**bsolute **s**hrinkage and **s**election **o**perator, deutsch: Operator der kleinsten Schrumpfung und Selektion) (Tibshirani 1996). Um zu sehen, warum L_1 zu einem dünn besetzten Problem führt, betrachten wir den Fall, dass es zwei Gewichte $[w_1, w_2]^T$ gibt (Figueiredo 2003): $[\|1, 0]^T\|_2 = \|[1/\sqrt{2}, 1/\sqrt{2}]^T\|_2 = 1$, wobei $\|[1, 0]^T\|_1 = 1 < \|[1/\sqrt{2}, 1/\sqrt{2}]^T\|_1 = \sqrt{2}$, d. h., die Norm L_1 präferiert es, w_2 gleich 0 zu setzen und ein großes w_2 zu verwenden, anstatt beide Werte klein zu halten.

LASSO

16.4.2 Regression mit Prior für die Präzision des Rauschens

Oben haben wir angenommen, dass β, die Präzision des Rauschens, bekannt ist und dass $\boldsymbol{w}$ der einzige Parameter ist, über den wir integrieren müssen. Wenn β nicht bekannt ist, können wir ebenfalls einen Prior definieren. Genau wie in Abschnitt 16.3 definieren wir einen Gamma-Prior

$$p(\beta) \sim \text{Gamma}(a_0, b_0)$$

und einen Prior für $\boldsymbol{w}$, bedingt unter β:

$$p(\boldsymbol{w}|\beta) \sim \mathcal{N}(\boldsymbol{\mu}_0, \beta\boldsymbol{\Sigma}_0) .$$

Im Falle $\boldsymbol{\mu}_0 = 0$ und $\boldsymbol{\Sigma}_0 = \alpha\mathbf{I}$ erhalten wir, wie oben diskutiert, Ridge-Regression. Wir können nun einen konjugierten Normal-Gamma-Prior für die Parameter $\boldsymbol{w}$ und β aufschreiben:

$$p(\boldsymbol{w}, \beta) = p(\beta)\, p(\boldsymbol{w}|\beta) \sim \text{Normal-Gamma}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0, a_0, b_0)$$

Man kann zeigen (Hoff 2009), dass der Posterior durch

$$p(\boldsymbol{w}, \beta|\mathbf{X}, \boldsymbol{r}) \sim \text{Normal-Gamma}(\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N, a_N, b_N)$$

gegeben ist, wobei

$$\begin{aligned}
\boldsymbol{\Sigma}_N &= \left(\mathbf{X}^T\mathbf{X} + \boldsymbol{\Sigma}_0\right)^{-1} , \\
\boldsymbol{\mu}_N &= \boldsymbol{\Sigma}_N\left(\mathbf{X}^T\boldsymbol{r} + \boldsymbol{\Sigma}_0\boldsymbol{\mu}_0\right) , \\
a_N &= a_0 + N/2 , \\
b_N &= b_0 + \frac{1}{2}\left(\boldsymbol{r}^T\boldsymbol{r} + \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}_0\boldsymbol{\mu}_0 - \boldsymbol{\mu}_N^T\boldsymbol{\Sigma}_N\boldsymbol{\mu}\right) .
\end{aligned} \tag{16.26}$$

Ein Beispiel ist in Abbildung 16.5 dargestellt. Dort werden Polynome unterschiedlicher Grade an eine kleine Menge von Instanzen angepasst – $\boldsymbol{w}$ entspricht dem Koeffizientenvektor des Polynoms. Wie wir sehen, führt die Maximum-Likelihood zur Überanpassung, wenn der Grad des Polynoms erhöht wird.

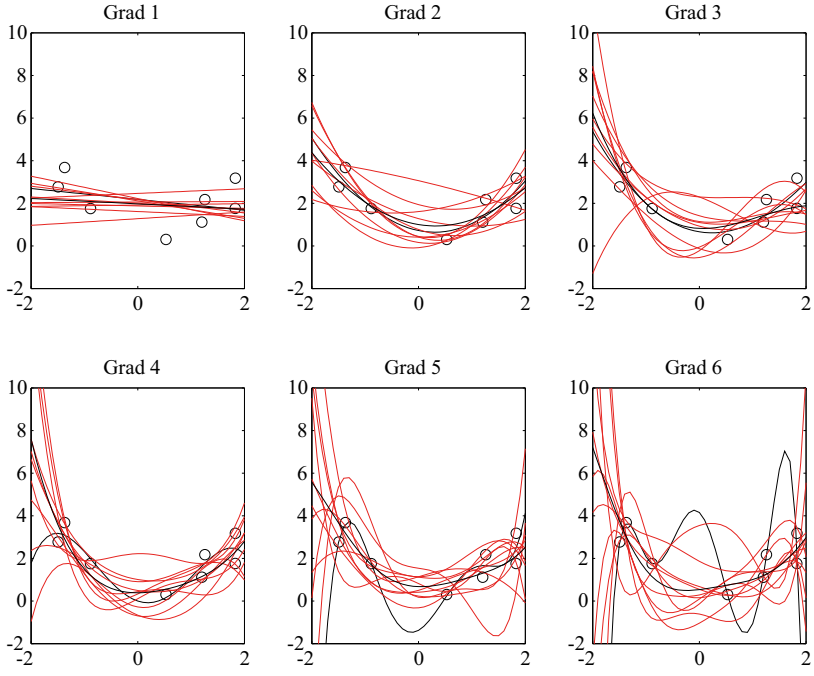


***Abb. 16.5*:** Beispiel für die Bayessche polynomiale Regression. Kreise stehen für die Datenpunkte, und die gestrichelte Kurve ist die Maximum-Likelihood-Anpassung, die zur Überanpassung führt, wenn der Grad des Polynoms größer wird. Die dünnen Kurven entsprechen zehn Stichproben aus dem Posterior $p(\boldsymbol{w}, \beta)$ und die dicke Kurve ist ihr Mittel.

Markov-Ketten-Monte-Carlo-Sampling

Wir verwenden das *Markov-Ketten-Monte-Carlo-Sampling,* um wie folgt die Bayessche Anpassung zu erhalten: Wir ziehen einen β-Wert aus $p(\gamma) \sim \text{Gamma}(a_N, b_N)$ und dann ziehen wir ein $\boldsymbol{w}$ aus $p(\boldsymbol{w}|\beta) \sim \mathcal{N}(\boldsymbol{\mu}_N, \beta\boldsymbol{\Sigma}_N)$, was uns ein Stichprobenmodell aus dem Posterior $p(\boldsymbol{w}, \beta)$ liefert. Für jeden Grad werden, wie in Abbildung 16.5 dargestellt, zehn solche Stichproben gezogen. Die dicke Kurve ist das Mittel dieser zehn Modelle und sie ist eine Approximation des vollständigen Integrals. Wir sehen, dass wir schon mit zehn Stichproben eine akzeptable und sehr glatte Anpassung an die Daten bekommen. Man beachte, dass keines der Stichprobenmodelle aus dem Posterior besser sein muss als der Maximum-Likelihood-Schätzer; es ist die Mittelung, die zu einer glatteren und damit besseren Anpassung führt.

16.4.3 Der Gebrauch von Basis/Kernel-Funktionen

Unter Verwendung des Bayesschen Schätzung gemäß Gleichung 16.23 kann die Vorhersage folgendermaßen geschrieben werden:

$$\begin{aligned} r' &= (\boldsymbol{x}')^T \boldsymbol{w} \\ &= \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \mathbf{X}^T \boldsymbol{r} \\ &= \sum_t \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \boldsymbol{x}^t r^t \,. \end{aligned}$$

DUALE DARSTELLUNG

Das ist die *duale Darstellung.* Wenn wir den Parameter wie bei Support-Vektor-Maschinen (Kapitel 13) mithilfe der Trainingsdaten oder einer Teilmenge von diesen ausdrücken können, dann können wir die Vorhersage als Funktion der aktuellen Eingabe und der vorherigen Daten schreiben. Wir formulieren daher die obige Schreibweise um zu

$$r' = \sum_t K(\boldsymbol{x}', \boldsymbol{x}^t) r^t \,, \tag{16.27}$$

wobei wir definieren

$$K(\boldsymbol{x}', \boldsymbol{x}^t) = \beta(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N \boldsymbol{x}^t \,. \tag{16.28}$$

BASISFUNKTION

Wie wir wissen, können wir den linearen Kernel aus Gleichung 16.28 verallgemeinern, indem wir eine nichtlineare *Basisfunktion* $\phi(\boldsymbol{x})$ verwenden. Mit dieser bilden wir in einen neuen Raum ab, in dem wir das lineare Modell anpassen. In diesem Fall haben wir anstelle des d-dimensionalen $\boldsymbol{x}$ das k-dimensionale $\boldsymbol{\phi}(\boldsymbol{x})$, wobei k die Anzahl der Basisfunktionen ist, und anstelle der $N \times d$-Matrix $\mathbf{X}$ haben wir ein $N \times k$-dimensionales Bild der Basisfunktionen $\boldsymbol{\Phi}$.

Während des Tests haben wir

$$\begin{aligned} r' &= \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{w} \text{ mit } \boldsymbol{w} = \beta \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\Phi}^T \boldsymbol{r} \text{ und } \boldsymbol{\Sigma}_N^{\phi} = (\alpha \mathbf{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \\ &= \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\Phi}^T \boldsymbol{r} \\ &= \sum_t \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\phi}(\boldsymbol{x}^t) r^t \\ &= \sum_t K(\boldsymbol{x}', \boldsymbol{x}^t) r^t \end{aligned} \tag{16.29}$$

und wir definieren

$$K(\boldsymbol{x}', \boldsymbol{x}^t) = \beta \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\phi}(\boldsymbol{x}^t) \tag{16.30}$$

als den äquivalenten Kernel. Dies ist die duale Darstellung im Raum der $\boldsymbol{\phi}$. Wir stellen fest, dass wir unsere Schätzung als gewichtete Summe

KERNEL-FUNKTION

der Effekte von Instanzen aus der Trainingsmenge schreiben können, wobei die Effekte durch die *Kernel-Funktion* $K(\boldsymbol{x}', \boldsymbol{x}^t)$ gegeben sind. Dies entspricht den nichtparametrischem Kernel-Glättern, die wir in Kapitel 8 diskutiert haben, oder auch den Kernel-Maschinen aus Kapitel 13.

Fehlerbalken können unter Verwendung von

$$\text{Var}(r') = \beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\Sigma}_N^{\phi} \boldsymbol{\phi}(\boldsymbol{x}')$$

definiert werden.

Beispiel für einen linearen und einen quadratischer Kernel sowie für einen Kernel sechsten Grades sind in Abbildung 16.6 zu sehen. Dies ist äquivalent mit der polynomialen Regression, die in Abbildung 16.5 illustriert ist, nur dass wir hier die duale Darstellung verwenden und die Polynomkoeffizienten $\boldsymbol{w}$ in die Kernel-Funktion eingebettet sind. Wie wir sehen, können wir, genau wie bei der eigentlichen Regression, wo wir mit den originalen $\boldsymbol{x}$ oder mit $\boldsymbol{\phi}(\boldsymbol{x})$ arbeiten können, auch bei der Bayesschen Regression die vorverarbeiteten $\boldsymbol{\phi}(\boldsymbol{x})$ verwenden, die Parameter in diesem Raum definieren. Weiter hinten in diesem Kapitel werden wir uns mit Gaußschen Prozessen beschäftigen, bei denen wir $K(\boldsymbol{x}, \boldsymbol{x}^t)$ direkt definieren und verwenden können, ohne $\boldsymbol{\phi}(\boldsymbol{x})$ berechnen zu müssen.

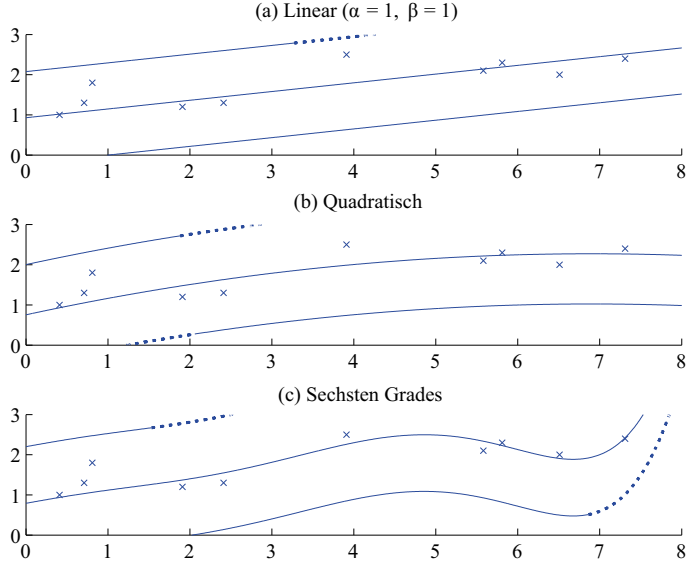


***Abb. 16.6*:** Bayessche Regression unter Verwendung eines Kernels mit Fehlerbalken von einer Standardabweichung: (a) linearer Kernel, $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x]^T$, (b) quadratischer Kernel, $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x, x^2]^T$ und (c) Kernel sechsten Grades, $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x, x^2, x^3, x^4, x^5, x^6]^T$.

16.4.4 Bayessche Klassifikation

Bei einem Zweiklassenproblem gibt es eine einzelne Ausgabe, und wenn wir ein lineares Modell annehmen, dann haben wir

$$P(C_1|\boldsymbol{x}^t) = y^t = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x}^t) .$$

Die Log-Likelihood einer Bernoulli-Stichprobe ist durch

$$\mathcal{L}(\boldsymbol{r}|\mathbf{X}) = \sum_t r^t \log y_t + (1-r^t)\log(1-y^t)$$

gegeben, was wir maximieren – bzw. wir minimieren ihren negativen Logarithmus, die Kreuzentropie –, um die ML-Schätzung zu finden, beispielsweise mittels Gradientenabstieg. Dies wird als *logistische Diskriminanz bezeichnet* (Abschnitt 10.7).

Beim Bayesschen Ansatz nehmen wir einen Gaußschen Prior an:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{m}_0, \mathbf{S}_0) \tag{16.31}$$

und der Logarithmus des Posteriors ist gegeben durch

$$\begin{aligned}\log p(\boldsymbol{w}|\boldsymbol{r},\mathbf{X}) &\propto \log p(\boldsymbol{w}) + \log p(\boldsymbol{r}|\boldsymbol{w},\mathbf{X}) \\ &= -\frac{1}{2}(\boldsymbol{w}-\boldsymbol{m}_0)^T\mathbf{S}_0^{-1}(\boldsymbol{w}-\boldsymbol{m}_0) \\ &\quad + \sum_t r^t \log y_t + (1-r^t)\log(1-y^t) + c\,. \end{aligned} \tag{16.32}$$

Diese a-posteriori-Verteilung ist nicht mehr Gaußsch und wir können nicht exakt integrieren. Wir können dann eine *Laplace-Approximation* verwenden, die folgendermaßen funktioniert (MacKay 2003). Angenommen, wir wollen eine Funktion $f(x)$ approximieren, die nicht notwendig normiert sein muss (d. h., ihr Integral muss nicht 1 sein). Bei der Laplace-Approximation ermitteln wir den Modalwert x_0 von $f(x)$, passen eine Gauß-Kurve $q(x)$ an, die um diesen zentriert ist und deren Kovarianz durch die Krümmung von $f(x)$ an diesem Mittelwert gegeben ist, und wenn wir $f(x)$ dann integrieren wollen, integrieren wir stattdessen diese angepasste Gauß-Kurve.

LAPLACE-APPROXIMATION

Um die Varianz der Gauß-Kurve zu finden, betrachten wir die Taylor-Entwicklung von $f(\cdot)$ an der Stelle $x = x_0$

$$\log f(x) = \log f(x_0) - \frac{1}{2}a(x-x_0)^2 + \ldots$$

mit

$$a \equiv -\frac{d}{dx^2}\log f(x)\bigg|_{x=x_0} .$$

Der erste, lineare Term verschwindet, weil die erste Ableitung im Modalwert 0 ist. Wir bilden den Exponenten und erhalten

$$f(x) = f(x_0) \exp\left[-\frac{a}{2}(x - x_0)^2\right] .$$

Um $f(x)$ zu normieren, beachten wir, dass für eine Gauß-Verteilung

$$\int \frac{1}{\sqrt{2\pi}(1/\sqrt{a})} \exp\left[-\frac{a}{2}(x - x_0)^2\right] = 1$$
$$\Rightarrow \int \exp\left[-\frac{a}{2}(x - x_0)^2\right] = \sqrt{a/2\pi}$$

gilt, und daher ist

$$q(x) = \sqrt{a/2\pi} \exp\left[-\frac{a}{2}(x - x_0)^2\right] \sim \mathcal{N}(x_0, 1/a) .$$

Im multivariaten Fall, in dem $\boldsymbol{x} \in \mathbb{R}^d$, haben wir

$$\log f(\boldsymbol{x}) = \log f(\boldsymbol{x}_0) - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^T \mathbf{A} (\boldsymbol{x} - \boldsymbol{x}_0) + \cdots ,$$

wobei $\mathbf{A}$ die (Hesse-)Matrix der zweiten Ableitungen ist:

$$\mathbf{A} = -\nabla\nabla \log f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0} .$$

Die Laplace-Approximation ist dann

$$f(\boldsymbol{x}) = \frac{|\mathbf{A}|^{1/2}}{(2\pi)^{d/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^T \mathbf{A} (\boldsymbol{x} - \boldsymbol{x}_0)\right] \sim \mathcal{N}_d(\boldsymbol{x}_0, \mathbf{A}^{-1}) .$$

Nachdem wir erläutert haben, wie wir unsere Funktion approximieren, können wir diese für die a-posteriori-Dichte verwenden. Die MAP-Schätzung $\boldsymbol{w}_{\text{MAP}}$ – der Modalwert von $p(\boldsymbol{w}|\boldsymbol{r}, \mathbf{X})$ – wird als der Mittelwert genommen, und die Kovarianzmatrix ist durch die Inverse der Matrix der zweiten Ableitungen der negativen Log-Likelihood gegeben:

$$\mathbf{S}_N = -\nabla\nabla \log p(\boldsymbol{w}|\boldsymbol{r}, \mathbf{X}) = \mathbf{S}_0^{-1} + \sum_t y^t (1 - y^t) \boldsymbol{x}^t (\boldsymbol{x}^t)^T .$$

Wir integrieren dann über diese Gauß-Funktion, um die Klassenwahrscheinlichkeit zu schätzen; wir erhalten

$$P(C_1|\boldsymbol{x}) = y = \int \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{x}) q(\boldsymbol{w})\, d\boldsymbol{w}$$

mit $q(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}_{\text{MAP}}, \mathbf{S}_N^{-1})$. Eine weitere Schwierigkeit ergibt sich daraus, dass wir über eine Gauß-Funktion, gefaltet mit einer Sigmoidfunktion, nicht analytisch integrieren können. Wenn wir stattdessen die *Probit-Funktion* verwenden, welche die gleiche S-Form hat wie die Sigmoidfunktion, dann ist eine analytische Lösung möglich (Bishop 2006).

PROBIT-FUNKTION

16.5 Wahl eines Priors

Die Definition des Priors ist der subjektive Teil bei der Bayesschen Schätzung und sollte mit großer Sorgfalt vorgenommen werden. Am besten ist es, robuste a-priori-Verteilungen mit starken Flanken zu definieren, um den Parameterraum nicht zu sehr einzuschränken. In dem Extremfall, dass es keine a-priori-Präferenz gibt, kann man einen uninformativen Prior verwenden. Hierfür sind spezielle Methoden vorgeschlagen worden, so zum Beispiel Jeffreys Prior (Murphy 2012). Manchmal ist unsere Wahl des Priors auch durch das Argument der Einfachheit motiviert – beispielsweise macht ein konjugierter Prior die Inferenz sehr einfach.

Eine kritische Entscheidung ist die, wann man für einen Parameter eine Konstante annehmen sollte und wann man ihn besser als Zufallsvariable mit einer a-priori-Verteilung definieren sollte, über die dann integriert (gemittelt) wird. In Abschnitt 16.4.1 haben wir zum Beispiel angenommen, dass wir die Präzision des Rauschens kennen; in Abschnitt 16.4.1 haben wir dagegen angenommen, dass wir sie nicht kennen, und haben stattdessen einen Gamma-Prior für sie definiert. Entsprechend nehmen wir für die Streuung der Gewichte bei der linearen Regression einen konstanten Wert α an, doch wir könnten auch eine a-priori-Verteilung für sie definieren und deren Mittelwert bestimmen, wenn wir wollen. Natürlich macht das den Prior und die gesamte Inferenz komplizierter, doch die Mittelung über α sollte vorgezogen werden, wenn wir nicht wissen, was ein guter Wert für α ist.

Eine weitere Entscheidung betrifft die Frage, wie weit wir bei der Definition des Priors gehen sollen. Angenommen, wir haben einen Parameter θ und definieren für diesen einen Posterior. Bei der Vorhersage haben wir

$$\text{Level I: } p(x|\mathcal{X}) = \int p(x|\theta)\, p(\theta|\mathcal{X})\, d\theta$$

mit $p(\theta|\mathcal{X}) \propto (\mathcal{X}|\theta)\, p(\theta)$. Wenn wir glauben, dass wir kein gutes $p(\theta)$ definieren können, sondern dass es von irgendeiner anderen Variable abhängt, dann können wir θ bedingt unter einem Hyperparameter α betrachten und über diesen integrieren:

$$\text{Level II: } p(x|\mathcal{X}) = \int p(x|\theta)\, p(\theta|\mathcal{X}, \alpha)\, p(\alpha)\, d\theta\, d\alpha\,.$$

Das nennt man einen *hierarchischen Prior*. Damit wird die Inferenz ziemlich schwierig, weil wir über zwei Levels integrieren müssen. Eine Abkürzung besteht darin, verschiedene Werte von α auf den Daten zu testen, das beste α^* zu wählen und genau diesen Wert zu verwenden:

$$\text{Level II ML: } p(x|\mathcal{X}) = \int p(x|\theta)\, p(\theta|\mathcal{X}, \alpha^*)\, d\theta\,.$$

Dies wird als *Level-II-Maximum-Likelihood* oder *empirischer Bayes-Prior* bezeichnet.

16.6 Bayesscher Modellvergleich

Angenommen, wir haben viele Modelle $\mathcal{M}_j$, von denen jedes seine eigene Parametermenge θ_j hat, und wir wollen diese Modelle vergleichen. In Abbildung 16.5 zum Beispiel haben wir Polynome unterschiedlicher Grade und wir wollen annehmen, dass wir prüfen wollen, wie gut sie die Daten anpassen.

Marginal-Likelihood

Für ein gegebenes Modell $\mathcal{M}$ und den Parameter θ ist die Likelihood der Daten $p(\mathcal{X}|\mathcal{M},\theta)$. Um für ein gegebenes Modell die Bayessche *Marginal-Likelihood* zu erhalten, mitteln wir über θ:

$$p(\mathcal{X}|\mathcal{M}) = \int p(\mathcal{X}|\theta,\mathcal{M})\, p(\theta|\mathcal{M})\, d\theta\,. \tag{16.33}$$

Dies wird als *Modellevidenz* bezeichnet. Beispielsweise gilt für den oben diskutierten Fall der polynomialen Regression

$$p(\boldsymbol{r}|\mathbf{X},\mathcal{M}) = \iint p(\boldsymbol{r}|\mathbf{X},\boldsymbol{w},\beta,\mathcal{M})\, p(\boldsymbol{w},\beta|\mathcal{M})\, d\boldsymbol{w}\, d\beta\,,$$

wobei $p(\boldsymbol{w},\beta|\mathcal{M})$ der für das Modell $\mathcal{M}$ angenommene Prior ist. Wir können dann für das Modell die a-posteriori-Wahrscheinlichkeit, gegeben die Daten, berechnen:

$$p(\mathcal{M}|\mathcal{X}) = \frac{p(\mathcal{X}|\mathcal{M})\, p(\mathcal{M})}{p(\mathcal{X})}\,. \tag{16.34}$$

Hierbei ist $P(\mathcal{M})$ die a-priori-Verteilung der Modelle. Eine komfortable Eigenheit des Bayesschen Ansatzes besteht darin, dass selbst wenn diese a-priori-Verteilungen als gleichmäßig angenommen werden, die Marginal-Likelihood einfachere Modelle favorisiert, da sie über alle θ mittelt. Nehmen wir nun an, wir haben Modelle zunehmender Komplexität, beispielsweise Polynome von wachsendem Grad.

Wir betrachten eine Datenmenge $\mathcal{X}$ mit N Instanzen. Ein komplexeres Modell wird in der Lage sein, eine größere Zahl solcher Datenmengen mit akzeptabler Güte anzupassen als ein einfacheres Modell. Betrachten wir die zufällige Wahl von drei Punkten in einer Ebene. Die Anzahl der Tripel, die durch eine Gerade angepasst werden können, ist viel kleiner als die Anzahl der Tripel, die durch eine quadratische Kurve angepasst werden können. Es gelte $\sum_{\mathcal{X}} p(\mathcal{X}|\mathcal{M}) = 1$. Da es für ein komplexes Modell mehr mögliche $\mathcal{X}$ gibt, für die es eine gute Anpassung machen kann, wird der Wert von $p(\mathcal{X}'|\mathcal{M})$ für ein spezielles $\mathcal{X}'$ kleiner – siehe Abbildung 16.7. Für ein einfacheres Modell liegt das Maximum von $p(\mathcal{M}|\mathcal{X})$ höher (selbst wenn wir annehmen, dass die a-priori-Verteilungen $p(\mathcal{M})$ gleich sind). Dies ist die Bayessche Interpretation von Ockams Rasiermesser (MacKay 2003).

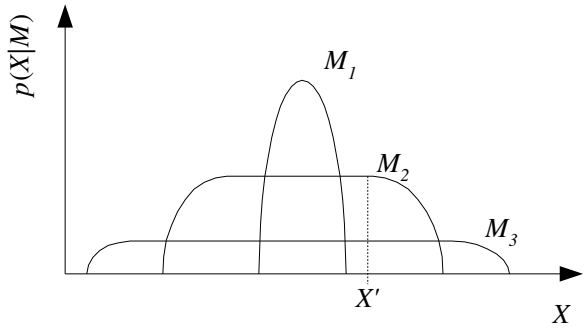


***Abb. 16.7*:** Der Bayessche Modellvergleich favorisiert einfachere Modelle. $\mathcal{M}_1$, $\mathcal{M}_2$ und $\mathcal{M}_3$ sind drei Modelle von zunehmender Komplexität. Die x-Achse ist der Raum aller Datenmengen mit N Instanzen. Ein komplexes Modell kann mehr Datenmengen anpassen, aber es breitet sich dünn über dem Raum aller möglichen Datenmengen der Größe N aus. Ein einfacheres Modell kann weniger Datenmengen anpassen, aber dabei hat jede eine größere Wahrscheinlichkeit. Für eine spezielle Datenmenge $\mathcal{X}'$, für die mit beiden Modellen die Anpassung möglich ist, wird das einfachere Modell die höherer Marginal-Likelihood haben (MacKay 2003).

Für das Beispiel der polynomialen Anpassung, das in Abbildung 16.5 dargestellt ist, ist ein Vergleich der Likelihood und der Marginal-Likelihood in Abbildung 16.8 gezeigt. Wie wir sehen, wächst die Likelihood, wenn die Komplexität zunimmt, was zur Überanpassung führt. Die Marginal-Likelihood wächst dagegen nur bis zu dem richtigen Grad des Polynoms und beginnt dann zu fallen. Das liegt daran, dass es viel mehr komplexe Modelle gibt, welche die Daten schlecht anpassen, und diese ziehen die Likelihood nach unten, wenn wir über die Modelle mitteln.

Wenn wir zwei Modelle $\mathcal{M}_0$ und $\mathcal{M}_1$ haben, können wir sie vergleichen:

$$\frac{P(\mathcal{M}_1|\mathcal{X})}{P(\mathcal{M}_0|\mathcal{X})} = \frac{P(\mathcal{X}|\mathcal{M}_1)}{P(\mathcal{X}|\mathcal{M}_0)} \frac{P(\mathcal{M}_1)}{P(\mathcal{M}_0)}$$

und wir haben größeres Vertrauen in $\mathcal{M}_1$, wenn dieses Verhältnis größer ist als 1; andernfalls haben wir größeres Vertrauen in $\mathcal{M}_0$.

An dieser Stelle sind zwei wichtige Anmerkungen zu machen. Erstens wird das Verhältnis der beiden Marginal-Likelihoods *Bayes-Faktor* genannt; er ist ausreichend für die Modellauswahl, selbst wenn die beiden a-priori-Verteilungen gleich sind. Zweitens führen wir beim Bayesschen Ansatz keine Modellauswahl durch; vielmehr entspricht es dem Bayesschen Ansatz, über die Vorhersagen der Modelle zu mitteln, anstatt eine auszuwählen und die anderen zu verwerfen. So ist es etwa bei dem oben diskutierten Beispiel der polynomialen Regression das Beste, ein

BAYES-FAKTOR

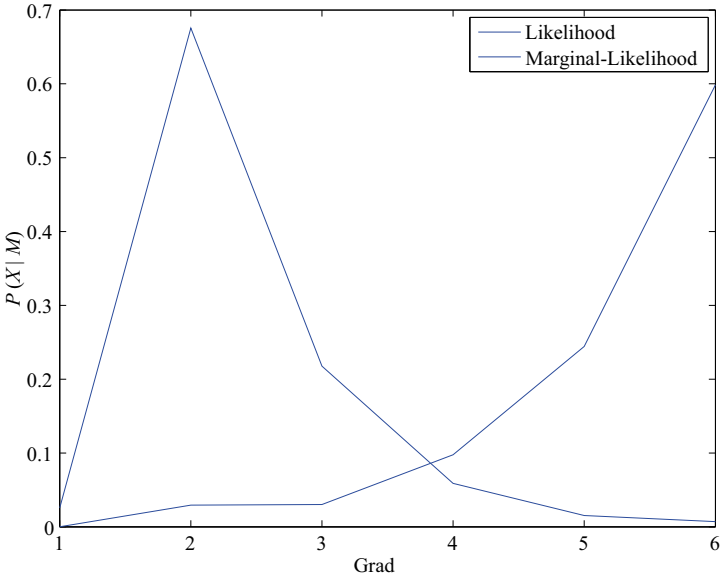


***Abb. 16.8*:** Likelihood vs. Marginal-Likelihood für das Beispiel der polynomialen Regression. Während die Likelihood mit dem Grad des Polynoms wächst, hat die Marginal-Likelihood, die über Parameterwerte mittelt, bei der richtigen Komplexität einen Peak und fällt dann ab.

gewichtetes Mittel über alle Grade zu nehmen (wobei die Gewichte die Marginal-Likelihoods sind), anstatt einen Grad auszuwählen.

BAYESSCHES INFORMATIONSKRITERIUM

Ein verwandter Ansatz ist das *Bayessche Informationskriterium* (Abk. BIC für engl. Bayesian information criterion), bei dem unter Verwendung der Laplace-Approximation (Abschnitt 16.4.4) Gleichung 16.33 durch

$$\log p(\mathcal{X}|\mathcal{M}) \approx \text{BIC} \equiv \log p(\mathcal{X}|\theta_{\text{ML}}, \mathcal{M}) - \frac{|\mathcal{M}|}{2} \log N \qquad (16.35)$$

genähert wird. Der erste Term ist die Likelihood unter Verwendung des ML-Schätzers und der zweite Term ist ein Strafterm für komplexe Modelle: $|\mathcal{M}|$ ist ein Maß für die Modellkomplexität, genauer gesagt für die Freiheitsgrade des Modells, und entspricht bei unserem Beispiel der polynomialen Regression der Anzahl der Koeffizienten. Mit wachsender Modellkomplexität kann der erste Term größer werden, was aber durch den zweiten Term kompensiert wird.

AKAIKES INFORMATIONSKRITERIUM

Ein weiterer verwandter, aber nicht-Bayesscher Ansatz ist *Akaikes Informationskriterium* (Abk. AIC für engl. Akaike's information criterion)

$$\text{AIC} \equiv \log p(\mathcal{X}|\theta_{\text{ML}}, \mathcal{M}) - |\mathcal{M}| \,. \qquad (16.36)$$

Auch hier haben wir einen Strafterm, der proportional zur Modellkomplexität ist. Es ist wichtig, darauf hinzuweisen, dass $|\mathcal{M}|$ bei solchen Kriterien den „effektiven“ Freiheitsgrad repräsentiert und nicht einfach die Anzahl der einstellbaren Parameter des Modells. Beispielsweise haben mehrlagige Perzeptronen (Kapitel 11) viel weniger effektive Freiheitsgrade als einstellbare Verbindungsgewichte.

Eine mögliche Interpretation des Strafterms besteht darin, ihn als Ausdruck für den „Optimismus“ anzusehen (Hastie, Tibshirani und Friedman 2011). In einem komplexen Modell würde der ML-Schätzer zur Überanpassung und somit zu einem sehr optimistischen Indikator für die Leistung des Modells führen; er sollte daher proportional zur Modellkomplexität gedämpft werden.

16.7 Bayessche Schätzung für ein Mischungsmodell

In Abschnitt 7.2 haben wir das Mischungsmodell diskutiert, bei dem wir die Dichte als eine gewichtete Summe der Dichten für die einzelnen Komponenten schreiben. Wir wiederholen Gleichung 7.1,

$$p(\boldsymbol{x}) = \sum_{i=1}^{k} P(\mathcal{G}_i)\, p(\boldsymbol{x}|\mathcal{G}_i) \,,$$

wobei die $P(\mathcal{G}_i)$ die Mischungsanteile sind und $p(\boldsymbol{x}|\mathcal{G}_i)$ die Dichten der Komponenten. Im Falle einer Gaußschen Mischung haben wir beispielsweise $p(\boldsymbol{x}|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, und mit der Definition $\pi_i \equiv P(\mathcal{G}_i)$ haben wir den Parametervektor $\Phi = \{\pi_i, \boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^k$, denn wir aus den Daten $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$ lernen müssen.

In Abschnitt 7.4 haben wir den EM-Algorithmus diskutiert, der eine Maximum-Likelihood-Prozedur ist:

$$\Phi_{\text{MLE}} = \arg\max_{\Phi} \log p(\mathcal{X}|\Phi) \,.$$

Wenn wir eine a-priori-Verteilung $p(\Phi)$ haben, dann können wir einen Bayesschen Ansatz konstruieren. Der MAP-Schätzer ist beispielsweise

$$\Phi_{\text{MAP}} = \arg\max_{\Phi} \log p(\Phi|\mathcal{X}) = \arg\max_{\Phi} \log p(\mathcal{X}|\Phi) + \log p(\Phi) \,. \tag{16.37}$$

Nun wollen wir den Prior aufschreiben. Die Π_i sind multinomiale Variablen, für die wir einen Dirichlet-Prior verwenden können (siehe Abschnitt 16.2.1). Für den Mittelwert und die Präzision (inverse Kovarianz)

der Gaußschen Komponenten können wir einen Normal-Wishart-Prior verwenden, wie wir in Abschnitt 16.3 gesehen haben:

$$\begin{aligned} p(\Phi) &= p(\boldsymbol{\pi}) \prod_i p(\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i) \\ &= \text{Dirichlet}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_i \text{Normal-Wishart}(\boldsymbol{\mu}_0, \kappa_0, v_0, \mathbf{V}_0) \; . \end{aligned} \tag{16.38}$$

Wenn wir den EM-Algorithmus für diesen Fall verwenden, ändert sich also der E-Schritt nicht, aber im M-Schritt maximieren wir den Posterior mit diesem Prior (Murphy 2012). Durch Addieren des Logarithmus des Posteriors wird Gleichung 7.10 zu

$$\begin{aligned} \mathcal{Q}(\Phi|\Phi^l) = &\sum_t \sum_i h_i^t \log \pi_i + \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi^l) + \log p(\pi) \\ &+ \sum_i \log p(\boldsymbol{\mu}_i, \boldsymbol{\Lambda}_i) \; , \end{aligned} \tag{16.39}$$

wobei die $h_i^t \equiv E[z_i^t]$ die im E-Schritt unter Verwendung der aktuellen Werte von Φ geschätzten weichen Labels sind. Die M-Schritt-MAP-Schätzung für die Mischungsanteile sind die folgenden (ausgehend von Gleichung 16.4):

$$\pi_i^{l+1} = \frac{\alpha_i + N_i - 1}{\sum_i \alpha_i + N - k} \tag{16.40}$$

mit $N_i = \sum_t h_i^t$. Die M-Schritt-MAP-Schätzungen für die Dichteparameter der Gaußschen Komponenten sind (ausgehend von Gleichung 16.16):

$$\begin{aligned} \boldsymbol{\mu}_i^{l+1} &= \frac{\kappa_0 \boldsymbol{\mu}_0 + N_i \boldsymbol{m}_i}{\kappa_0 + N_i} \; , \\ \boldsymbol{\Lambda}_i^{l+1} &= \left(\frac{\mathbf{V}_0^{-1} + \mathbf{C}_i + \mathbf{S}_i}{v_0 + N_i + d + 2} \right)^{-1} \; . \end{aligned} \tag{16.41}$$

Dabei sind die $\boldsymbol{m}_i = \sum_t h_i^t / N_i$ die Mittelwerte der Komponenten, $\mathbf{C}_i = \sum_t h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T$ ist die innere Streuungsmatrix der Komponente i und $\mathbf{S}_i = (\kappa_0 N_i)/(\kappa_0 + N_i)(\boldsymbol{m}_i - \boldsymbol{\mu}_0)(\boldsymbol{m}_i - \boldsymbol{\mu}_0)^T$ ist die Streuungsmatrix der Komponente i um den Prior-Mittelwert.

Wenn wir $\alpha_i = 1/K$ annehmen, ist dies ein gleichmäßiger Prior. Wir können $\kappa_0 = 0$ setzen, um die Mittelwertschätzungen nicht zu verzerren, es sei denn, wir haben irgendwelche a-priori-Informationen über sie. Für $\mathbf{V}_0$ können wir die Einheitsmatrix nehmen, so dass die MAP-Schätzung folglich einen regularisierenden Effekt hat.

Die Mischungsdichte ist als generatives Graphenmodell in Abbildung 16.9 dargestellt.

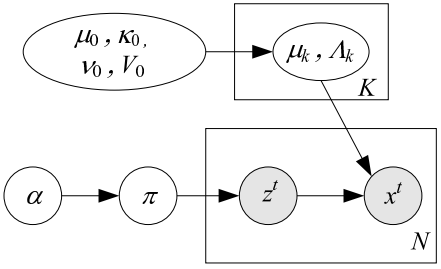


***Abb. 16.9*:** Die generative Graphenrepräsentation für ein Gaußsches Mischungsmodell.

Nachdem wir wissen, wie wir die Grundbausteine in Bayesscher Weise konstruieren, können wir sie zu komplizierteren Modellen kombinieren. Beispielsweise erhalten wir durch die Kombination des hier diskutierten Mischungsmodells mit dem linearen Regressionsmodell aus Abschnitt 16.4.1 eine Bayessche Version von gemischten Expertenmodellen (Abschnitt 12.8), bei der wir die Daten in Cluster aufteilen und simultan für jedes Cluster ein separates lineares Regressionsmodell lernen. Der Posterior erweist sich als ziemlich bösartig. Waterhouse et al. 1996 verwenden die Variationsapproximation, die grob gesagt wie folgt funktioniert.

Wir erinnern uns, dass wir $p(\theta|\mathcal{X})$ bei der Laplace-Approximation durch eine Gauß-Kurve nähern. Bei der *Variationsapproximation* nähern wir den Posterior durch eine Dichte $q(\mathcal{Z}|\psi)$, deren Parameter ψ einstellbar sind (Jordan et al. 1999; MacKay 2003; Bishop 2006). Diese Methode ist folglich allgemeiner, da wir nicht darauf festgelegt sind, eine Gaußsche Dichte zu verwenden. $\mathcal{Z}$ umfasst dabei alle latenten Variablen des Modells *und* die Parameter θ; die ψ des approximierenden Modells $q(\mathcal{Z}|\psi)$ werden so eingestellt, dass $q(\mathcal{Z}|\psi)$ so nahe wie möglich bei $p(\mathcal{Z}|\mathcal{X})$ liegt.

VARIATIONS-APPROXIMATION

Wir definieren den *Kullback-Leibler-Abstand* zwischen den beiden:

KULLBACK-LEIBLER-ABSTAND

$$D_{\mathrm{KL}}(q||p) = \sum_{\mathcal{Z}} q(\mathcal{Z}|\psi) \log \frac{q(\mathcal{Z}|\psi)}{q(\mathcal{Z}|\mathcal{X})} \,. \tag{16.42}$$

Um uns das Leben etwas einfacher zu machen, nehmen wir an, dass die Menge der latenten Variablen (einschließlich der Parameter) in Teilmengen $\mathcal{Z}_i, i = 1, \ldots, k$ partitioniert ist, so dass die Variationsverteilung faktorisiert werden kann:

$$q(\mathcal{Z}|\psi) = \prod_{i=1}^{k} q_i(\mathcal{Z}_i|\psi_i) \,. \tag{16.43}$$

Die Einstellung der Parameter ψ_i für jeden Faktor ist iterativ, recht ähnlich wie beim Expectation-Maximation-Algorithmus, den wir in Abschnitt 7.4 diskutiert haben. Wir starten mit (eventuell zufälligen) Anfangswerten, und beim Einstellen der einzelnen Parameter verwenden wir

Mean-Field-Approximation

die Erwartungswerte der $\mathbf{Z}_j, j \neq i$, wobei wir ringförmig vorgehen. Diese Methode wird als *Mean-Field-Approximation* bezeichnet.

Diese Faktorisierung ist eine Näherung. Beispielsweise schreiben wir in Abschnitt 16.4.2 bei der Behandlung der Regression

$$p(\boldsymbol{w}, \beta) = p(\beta)\, p(\boldsymbol{w}|\beta) \,,$$

da $\boldsymbol{w}$ bedingt unter β ist. Bei einer Variationsapproximation würden wir annehmen

$$p(\boldsymbol{w}, \beta) = p(\beta)\, p(\boldsymbol{w}) \,.$$

Bei gemischten Expertenmodellen zum Beispiel sind die latenten Parameter die Komponentenindizes und die Parameter sind die Parameter des Gattermodells, die Regressionsgewichte in den lokalen Experten, die Varianz des Rauschens und die Hyperparameter der a-priori-Verteilungen für das Gatter und die Regressionsgewichte: sie alle sind Faktoren (Waterhouse, MacKay und Robinson 1996).

16.8 Nichtparametrische Bayessche Modelle

Die Modelle, die wir weiter vorn in diesem Kapitel diskutiert haben, sind alle parametrisch in dem Sinne, dass wir Modelle konstanter Komplexität mit einer Menge von Parametern haben, und diese Parameter werden unter Verwendung der Daten und der a-priori-Information optimiert. In Kapitel 8 haben wir uns mit nichtparametrischen Modellen beschäftigt, in denen die Trainingsdaten das Modell bilden, so dass die Modellkomplexität vom Umfang der Daten abhängt. Wir wollen uns nun der Frage widmen, wie ein solcher Ansatz bei der Bayesschen Herangehensweise benutzt werden kann.

Die Bezeichnung nichtparametrisches Modell bedeutet nicht, dass das Modell keine Parameter hat. Vielmehr bezieht sie sich darauf, dass die Anzahl der Parameter nicht festgelegt ist, sondern in Abhängigkeit vom Umfang der Daten – oder besser gesagt von der Komplexität der den Daten innewohnenden Ordnung – wachsen kann. Solche Modelle werden manchmal *infinit* genannt, was sich auf die Eigenschaft bezieht, dass ihre Komplexität immer weiter anwachsen kann, wenn mehr Daten hinzukommen. In Abschnitt 11.9 haben wir inkrementelle neuronale Netze behandelt, bei denen wenn nötig neue verborgene Einheiten hinzugefügt wurden, so dass das Netz während des Trainings größer wurde. Beim parametrischen Lernen wird das Einstellen der Modellkomplexität jedoch üblicherweise in einer äußeren Schleife vorgenommen, indem die Leistung auf einer separaten Validierungsmenge geprüft wird. Der nichtparametrische Bayessche Ansatz umfasst die Modelleinstellung beim Training der

Parameter mithilfe eines geeigneten Priors (Gershman und Blei 2012). Das macht solche Modelle flexibler, und es würde sie normalerweise für Überanpassung empfänglich machen, wenn nicht der Bayessche Ansatz an sich dieses Risiko verringern würde.

Weil es die Parameter sind, die wachsen, sollten die a-priori-Verteilungen für solche Parameter in der Lage sein, mit diesem Wachstum umzugehen. Wir werden im Folgenden drei Beispielverteilungen für drei verschiedene Anwendungsfelder des maschinellen Lernens betrachten, und zwar Gaußsche Prozesse für das überwachte Lernen, Dirichlet-Prozesse für die Clusteranalyse und Betaprozesse für die Dimensionalitätsreduktion.

16.9 Gaußsche Prozesse

Betrachten wir das lineare Modell $y = \boldsymbol{w}^T\boldsymbol{x}$. Wir haben dann für jedes $\boldsymbol{w}$ eine Linie. Für eine gegebene a-priori-Verteilung $p(\boldsymbol{w})$ erhalten wir eine Verteilung von Linien oder genauer gesagt, für jedes $\boldsymbol{w}$ erhalten wir eine Verteilung von y-Werten, berechnet bei x zu $x(\boldsymbol{x}|\boldsymbol{w})$, wobei $\boldsymbol{w}$ aus $p(\boldsymbol{w})$ gezogen wird. Dies ist es, was wir unter einem *Gaußschen Prozess* verstehen wollen. Wir wissen, dass für ein Gaußsches $p(\boldsymbol{w})$ jedes y eine Linearkombination von Gauß-Variablen ist und als solche ebenfalls eine Gauß-Variable. Insbesondere interessieren wir uns für die Verbundverteilung der y-Werte, die aus den N Eingabedatenpunkten $\boldsymbol{x}^t, t = 1, \ldots, N$ berechnet wird (MacKay 1998).

GAUSSSCHER PROZESS

Wir nehmen einen Gauß-Prior mit Mittelwert null an:

$$p(\boldsymbol{w}) \sim \mathcal{N}(\mathbf{0}, (1/\alpha)\mathbf{I}) .$$

Wenn die $N \times d$ Datenpunkte $\mathbf{X}$ und der Gewichtsvektor der Länge d gegeben sind, schreiben wir die Ausgaben y in der Form

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{w} . \tag{16.44}$$

Dies ist eine N-variate Gauß-Variable mit

$$\begin{aligned} E[\boldsymbol{y}] &= \mathbf{X}E[\boldsymbol{w}] = \mathbf{0} , \\ \mathrm{Cov}(\boldsymbol{y}) &= E[\boldsymbol{y}\boldsymbol{y}^T] = \mathbf{X}E[\boldsymbol{w}\boldsymbol{w}^T]\mathbf{X}^T = \frac{1}{\alpha}\mathbf{X}\mathbf{X}^T \equiv \mathbf{K} . \end{aligned} \tag{16.45}$$

Dabei ist $\mathbf{K}$ die (Gram-)Matrix mit den Elementen

$$K_{i,j} \equiv K(\boldsymbol{x}^i, \boldsymbol{x}^j) = \frac{(\boldsymbol{x}^i)^T\boldsymbol{x}^j}{\alpha} .$$

Diese ist in der Literatur über Gauß-Prozesse als *Kovarianzfunktion* bekannt, und die Idee dahinter ist die gleiche wie bei Kernel-Funktionen:

KOVARIANZFUNKTION

Wenn wir einen Satz von Basisfunktionen $\boldsymbol{\phi}(\boldsymbol{x})$ verwenden, dann verallgemeinern wir das Konzept des Skalarprodukts der Originaleingaben zu einem Skalarprodukt von Basisfunktionen, das durch einen Kernel beschrieben wird:

$$K_{i,j} = \frac{\boldsymbol{\phi}(\boldsymbol{x}^i)^T \boldsymbol{\phi}(\boldsymbol{x}^j)}{\alpha} .$$

Die tatsächlich beobachtete Ausgabe r ist gegeben durch die Linie zuzüglich eines additiven Rauschens, $r = y + \epsilon$ mit $\epsilon \sim \mathcal{N}(0, \beta^{-1})$. Für alle N Datenpunkte schreiben wir die Ausgabe in der Form

$$\boldsymbol{r} \sim \mathcal{N}_N(\mathbf{0}, \mathbf{C}_N) \text{ mit } \mathbf{C}_N = \beta^{-1}\mathbf{I} + \mathbf{K} . \tag{16.46}$$

Um eine Vorhersage zu machen, betrachten wir den neuen Datensatz als das $(N+1)$-te Datenpunktpaar $(\boldsymbol{x}', r')$ und schreiben die Ausgabe unter Verwendung aller $N+1$ Datenpunkte. Wir haben

$$\boldsymbol{r}_{N+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{N+1}) \tag{16.47}$$

mit

$$\mathbf{C}_{N+1} = \begin{bmatrix} \mathbf{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{bmatrix} .$$

Dabei ist $\boldsymbol{k}$ der $N+1$-dimensionale Vektor der $K(\boldsymbol{x}', \boldsymbol{x}^t), t = 1, \ldots, N$ und $c = K(\boldsymbol{x}', \boldsymbol{x}') + \beta^{-1}$. Um eine Vorhersage zu machen, berechnen wir dann $p(r'|\boldsymbol{x}', \mathbf{X}, \boldsymbol{r})$; das ist eine Gauß-Variable mit

$$\begin{aligned} E[r'|\boldsymbol{x}'] &= \boldsymbol{k}^T \mathbf{C}_N^{-1} \boldsymbol{r} , \\ \mathrm{Var}(r'|\boldsymbol{x}') &= c - \boldsymbol{k}^T \mathbf{C}_N^{-1} \boldsymbol{k} . \end{aligned}$$

Das in Abbildung 16.10 gezeigte Beispiel verwendet einen linearen, einen quadratischen und einen Gaußschen Kernel. Die ersten beiden werden durch das Skalarprodukt ihrer zugehörigen Basisfunktionen definiert, und der Gaußsche Kernel wird direkt definiert durch

$$K_G(\boldsymbol{x}^i, \boldsymbol{x}^j) = \exp\left[-\frac{\|\boldsymbol{x}^i - \boldsymbol{x}^j\|^2}{s^2}\right] .$$

Der Mittelwert, der unsere Punktschätzung ist (wenn wir nicht über die vollständige Verteilung integrieren), kann auch als gewichtete Summe der Kernel-Effekte geschrieben werden:

$$E[r'|\boldsymbol{x}'] = \sum_t a^t K(\boldsymbol{x}^t, \boldsymbol{x}') . \tag{16.48}$$

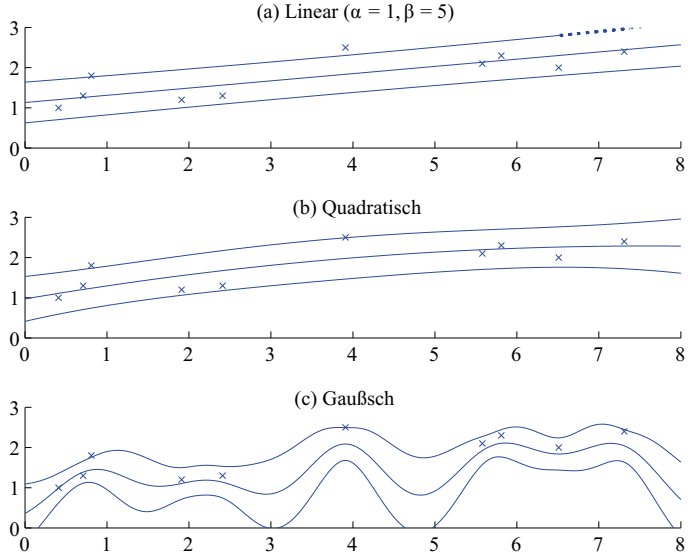


***Abb. 16.10*:** Regression mit Gaußschen Prozessen mit Fehlerbalken von einer Standardabweichung: (a) linearer Kernel, (b) quadratischer Kernel, (c) Gaußscher Kernel mit $s^2 = 0{,}5$.

Dabei ist a^t die t-te Komponente von $\mathbf{C}_N^{-1}\boldsymbol{r}$. Alternativ können wir dies als eine gewichtete Summe der Ausgaben der Trainingsdatenpunkte schreiben, wobei die Gewichte durch die Kernel-Funktion

$$E[r'|\boldsymbol{x}'] = \sum_t r^t w^t \tag{16.49}$$

gegeben sind, wobei w^t die t-te Komponente von $\boldsymbol{k}^T\mathbf{C}_N^{-1}$ bezeichnet.

Beachten Sie, dass wir auch die Varianz einer Vorhersage in einem Punkt berechnen könnten, um eine Vorstellung von der dort bestehenden Unsicherheit zu bekommen. Im Falle eines Gaußschen Kernels haben nur Instanzen innerhalb einer Umgebung Einfluss und die Varianz der Vorhersage ist hoch, wenn es in dieser Umgebung nur wenige Instanzen gibt (siehe Abbildung 16.11).

Kernel-Funktionen können für alle Anwendungen definiert und benutzt werden, wie wir zuvor in Kapitel 13 im Zusammenhang mit Kernel-Maschinen erläutert haben. Die Möglichkeit, Kernel-Funktionen direkt zu nutzen, ohne die Basisfunktionen berechnen oder speichern zu müssen, eröffnet uns eine große Flexibilität. Normalerweise berechnen wir, wenn wir eine Trainingsmenge gegeben haben, zunächst die Parameter, wobei wir zum Beispiel Gleichung 16.21 verwenden. Dann machen wir mithilfe dieser Parameter und Gleichung 16.22 Vorhersagen, wozu wir die

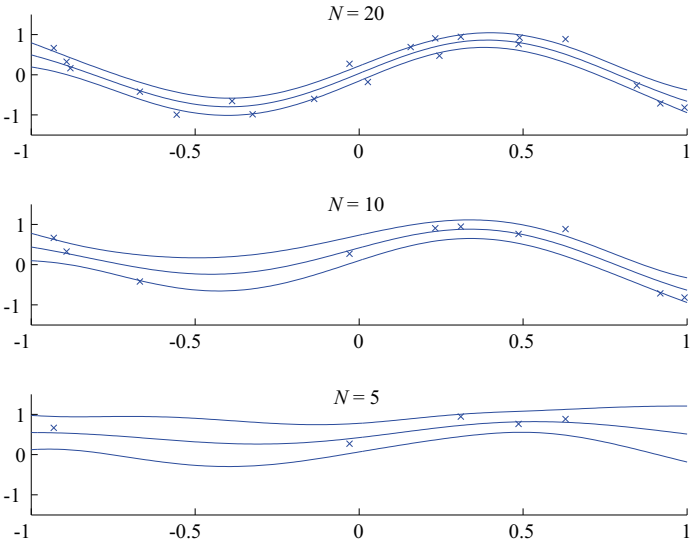


Abb. 16.11: Regression mit einem Gaußschen Kernel mit $s^2 = 0{,}5$ und unterschiedlich großen Datenmengen. Wie wir sehen, wird die Varianz der Vorhersage größer, wo es nur wenige Daten gibt.

Trainingsmenge nicht mehr brauchen. Dies ist sinnvoll, weil die Dimensionalität der Parameter von der Ordnung $\mathcal{O}(d)$ und damit im Allgemeinen kleiner als die Größe N der Trainingsmenge ist.

Wenn wir mit Basisfunktionen arbeiten, werden wir die Parameter allerdings nicht mehr explizit berechnen, weil die Dimensionalität der Basisfunktionen sehr hoch sein kann, ja sogar unendlich. In einem solchen Fall ist es günstiger, die duale Darstellung zu verwenden, wobei die Effekte der Trainingsinstanzen, die Kernel-Funktionen verwenden, berücksichtigt werden – also das, was wir hier tun. Diese Idee wird auch bei nichtparametrischen Glättern (Kapitel 8) und Kernel-Maschinen (Kapitel 13) genutzt.

Die Forderung ist hier, dass $\mathbf{C}_N$ invertierbar und somit positiv definit sein soll. Um das zu erfüllen, muss $\mathbf{K}$ semidefinit sein, so dass wir nach dem Addieren von β^{-1} zur Diagonale positive Definitheit erhalten. Wir stellen außerdem fest, dass die teuerste Operation das Invertieren dieser $N \times N$-Matrix ist, doch zum Glück muss die Inverse nur einmal berechnet und gespeichert werden (während des Trainings). Dennoch kann für große N eine Approximation notwendig sein.

Wird die Ausgabe für die Klassifikation (Zweiklassenproblem) verwendet, dann wird sie durch eine Sigmoidfunktion, $y = (\boldsymbol{w}^T \boldsymbol{x})$, gefiltert. Die Verteilung von y ist dann nicht mehr Gaußsch. Die Herleitung ist ähnlich,

außer dass die bedingte Verteilung $p(r_{N+1}|\boldsymbol{x}_{N+1}, \mathbf{X}, \boldsymbol{r})$ ebenfalls nicht mehr Gaußsch ist und wir nähern müssen, wozu wir beispielsweise die Laplace-Approximation verwenden können (Bishop 2006; Rasmussen und Williams 2006).

16.10 Dirichlet-Prozesse und Chinaestaurants

Um zu erklären, was ein Dirichlet-Prozess, beginnen wir mit einer Metapher: Wir stellen uns ein Chinarestaurant mit vielen Tischen vor. Die Gäste betreten das Restaurant, und zwar einer nach dem anderen. Der erste Gast setzt sich an Tisch eins. Jeder nachfolgende Gast kann sich entweder an einen der schon teilweise besetzten Tische dazusetzen oder einen neuen Tisch beginnen. Die Wahrscheinlichkeit, dass sich ein Gast an einen teilweise besetzten Tisch dazusetzt, ist proportional zu der Anzahl der bereits dort sitzenden Gäste, und die Wahrscheinlichkeit, dass er einen neuen Tisch beginnt, hängt von einem Parameter α ab. Dieser Prozess wird *Chinarestaurant-Prozess* genannt:

CHINARESTAURANT-PROZESS

dazusetzen an Tisch i mit $P(z_i = 1) = \dfrac{n_i}{\alpha + n - 1},\ i = 1, \ldots, k\,,$

an neuen Tisch setzen mit $P(z_{k+1} = 1) = \dfrac{\alpha}{\alpha + n - 1}\,.$

Dabei ist n_i die Anzahl der Gäste, die schon an Tisch i sitzen, und $n = \sum_{i=1}^{k} n_i$ ist die Gesamtzahl der Gäste. α ist die Neigung, einen neuen Tisch anzufangen; dies ist der Parameter in diesem Prozess. Man beachte, dass die Sitzanordnung der Gäste in jedem Schritt eine Partition der natürlichen Zahlen 1 bis n in k Teilmengen definiert. Dies nennt man einen *Dirichlet-Prozess* mit dem Parameter α.

DIRICHLET-PROZESS

Dieses Konzept können wir auf das Clustern anwenden, indem wir die Entscheidungen der Gäste nicht nur von den Besetzungszahlen der Tische abhängen lassen, sondern außerdem von der Eingabe. Nehmen wir an, es handelt sich nicht um ein Chinarestaurant, sondern um das Dinner auf einer großen Konferenz. Es gibt einen großen Speisesaal mit vielen Tischen und am Abend betreten die Konferenzteilnehmer einer nach dem anderen den Saal. Sie wollen natürlich essen, aber außerdem möchten sie gern interessante Gespräche führen. Deshalb wählen sie vorzugsweise einen Tisch, an dem bereits viele Leute sitzen, wobei sie darauf achten, in der Nähe von Kollegen zu sitzen, die ähnliche Forschungsinteressen haben. Wenn sie keinen solchen Tisch sehen, beginnen sie einen neuen Tisch in der Erwartung, dass sie dort von neu eintreffenden Konferenzteilnehmern entdeckt werden, die sich zu ihnen setzen.

Nehmen wir an, dass die Instanz (der Teilnehmer) t durch einen d-dimensionalen Vektor $\boldsymbol{x}^t$ repräsentiert wird und diese $\boldsymbol{x}^t$ lokal gaußverteilt

sind. Dies definiert eine Gaußsche Mischung über den gesamten Raum (den Speisesaal), und damit diese Bayessch wird, definieren wir wie in Abschnitt 16.7 a-priori-Verteilungen für die Parameter der Gauß-Komponenten. Um sie nichtparametrisch zu machen, definieren wir einen Dirichlet-Prozess als Prior, so dass eine neue Komponente bei Bedarf wie folgt hinzugefügt werden kann:

$$\text{teile Komponente } i \ (i = 1, \ldots, k) \text{ mit } P(z_i^t) \propto \frac{n_i}{\alpha + n - 1} p(\boldsymbol{x}^t | \mathcal{X}_i) \,,$$
$$\text{beginne neue Komponente mit } P(z_{k+1}^t = 1) \propto \frac{\alpha}{\alpha + n - 1} p(\boldsymbol{x}^t) \,.$$

$\mathcal{X}_i$ ist die Menge der Instanzen, die zuvor der Komponente i zugewiesen wurden. Indem wir ihre Daten und die a-priori-Verteilung verwenden, können wir einen Posterior berechnen, und indem wir über diesen integrieren, können wir $p(\boldsymbol{x}^t|\mathcal{X}_i)$ berechnen. Grob gesagt ist die Wahrscheinlichkeit, dass diese neue Instanz der Komponente i zugewiesen wird, hoch, wenn es bereits viele Instanzen in der Komponente gibt, also aufgrund eines hohen Priors, oder wenn $\boldsymbol{x}^t$ den Instanzen ähnelt, die bereits in $\mathcal{X}_i$ sind. Wenn keine der existierenden Komponenten eine hohe Wahrscheinlichkeit hat, wird eine neue Komponente hinzugefügt: $p(\boldsymbol{x}^t)$ ist die Randwahrscheinlichkeit (integriert über die a-priori-Verteilungen der Komponentenparameter, da es keine Daten gibt).

Unterschiedliche α können zu unterschiedlich vielen Clustern führen. Um den Parameter α einzustellen, können wir einen empirischen Bayes-Prior verwenden oder auch einen Prior für den Parameter definieren und ihn durch Mittelung bestimmen.

In Kapitel 7 haben wir im Zusammenhang mit der k-Means-Clustermethode (Abschnitt 7.3) den Leader-Cluster-Algorithmus diskutiert, bei dem neue Cluster während des Trainings hinzugefügt werden, und als Beispiel hierfür haben wir in Abschnitt 12.2.2 die adaptive Resonanztheorie vorgestellt, bei der ein neues Cluster dann hinzugefügt wird, wenn der Abstand zum Mittelpunkt des nächsten Clusters größer ist als ein Schwellwert (Vigilanz). Der Fall, mit dem wir uns hier beschäftigen, ist ähnlich: Gaußsche Komponenten und diagonale Kovarianzmatrizen vorausgesetzt, bedeutet ein großer Euklidischer Abstand zu allen Clustern, dass alle a-posteriori-Wahrscheinlichkeiten klein sind und eine neue Komponente hinzugefügt wird.

16.11 Latente Dirichlet-Allokation

Themenmodellierung

Betrachten wir nun eine Anwendung des Bayesschen Ansatzes bei der Textverarbeitung, und zwar die *Themenmodellierung* (Blei 2012). Heutzutage gibt es digitale Repositorys, die eine große Anzahl von Dokumenten enthalten – wissenschaftliche Artikel, Webseiten, E-Mails, Blogeinträge

usw. – doch es ist sehr schwierig, für eine Suchanfrage das passende Thema zu finden, es sei denn, den Dokumenten wurden manuell Themen wie „Kunst“, „Sport“ usw. zugeordnet. Hier würden wir uns wünschen, dass diese Zuordnung automatisch erfolgt.

Angenommen, wir haben ein Vokabular von M Wörtern. Jedes Dokument enthält N Wörter, die in unterschiedlichen Anteilen aus einer Reihe von Themen ausgewählt wurden – d. h., jedes Dokument ist eine Wahrscheinlichkeitsverteilung auf den Themen. Ein Dokument kann teilweise zu „Kunst“, teilweise zu „Sport“ und teilweise zu weiteren gehören. Jedes Thema ist seinerseits durch eine Mischung aus den M Wörtern definiert – d. h., jedes Thema entspricht einer Wahrscheinlichkeitsverteilung auf den Wörtern. Für das Thema „Kunst“ zum Beispiel haben die Wörter „Gemälde“ und „Skulptur“ eine hohe Wahrscheinlichkeit, das Wort „Knie“ dagegen nur eine kleine.

latente Dirichlet-Allokation

Bei der *latenten Dirichlet-Allokation* definieren wir einen generativen Prozess wie folgt (siehe Abbildung 16.12): Es gibt K Themen, ein Vokabular von M Wörtern und alle Dokumente enthalten N Wörter (Blei, Ng und Jordan 2003).

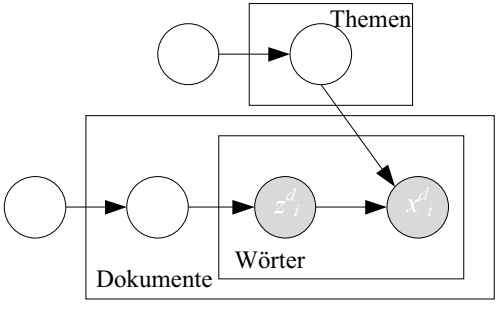


Abb. 16.12: Das Graphenmodell für die latente Dirichlet-Allokation.

Um das Dokument d zu generieren, entscheiden wir zunächst über die Themen, von denen es handelt. Diese Themenwahrscheinlichkeiten, $\pi^d_k, k = 1, \ldots, K$, definieren eine multinomiale Verteilung, und sie werden aus einem Dirichlet-Prior mit dem Hyperparameter $\boldsymbol{\alpha}$ gezogen (Abschnitt 16.2.1):

$$\boldsymbol{\pi}^d \sim \text{Dirichlet}_K(\boldsymbol{\alpha}) .$$

Nachdem wir die Themenverteilung für das Dokument d kennen, generieren wir die N Wörter, die es verwendet. Beim Generieren von Wort i entscheiden wir zunächst, aus welchem Thema es sein soll, indem wir eine Stichprobe aus $\boldsymbol{\pi}$ ziehen: Wir „würfeln“, wobei unser Würfel K Flächen hat, und die Fläche k hat die Wahrscheinlichkeit π_k. Wir definieren z^d_i als den Ausgang des Würfelns, der ein Wert zwischen 1 und K sein kann:

$$z^d_i \sim \text{Mult}_K(\boldsymbol{\pi}^d) .$$

Nun wissen wir, dass das i-te Wort in Dokument d zum Thema $z_i^d \in \{1, \ldots, K\}$ gehört. Wir haben eine $K \times M$-Matrix von Wahrscheinlichkeiten $\mathbf{W}$, deren Zeile k, $\boldsymbol{w}_k \equiv [w_{k1}, \ldots, w_{kM}]^T$, die Auftrittswahrscheinlichkeiten der M Wörter in Thema k angibt. Wir wissen also, dass das Thema von Wort i aus z_i^d stammen muss und ziehen eine Stichprobe aus der Multinomialverteilung, deren Parameter durch die Zeile z_i^d von $\mathbf{W}$ gegeben sind, um das Wort x_i^d zu erhalten (was ein Wert zwischen 1 und M ist):

$$x_i^d \sim \text{Mult}_M \left(\boldsymbol{w}_{z_i^d} \right) .$$

Dies ist eine multinomiale Stichprobenziehung, und wir definieren einen Dirichlet-Prior mit dem Hyperparameter $\boldsymbol{\beta}$ auf diesen Zeilen von multinomialen Wahrscheinlichkeiten:

$$\boldsymbol{w}_k \sim \text{Dirichlet}(\boldsymbol{\beta}) .$$

Damit ist der Prozess der Generierung eines Wortes abgeschlossen. Um alle N Wörter für das Dokument zu generieren, wiederholen wir den Prozess N-mal; d. h., für jedes Wort entscheiden wir über das Thema und wählen dann bei gegebenem Thema ein Wort (innere Tafel in der Abbildung). Wenn wir zum nächsten Dokument übergehen, ziehen wir zuerst eine neue Themenverteilung $\boldsymbol{\pi}$ (äußere Tafel) und dann die N Wörter aus dieser Themenverteilung.

Für alle Dokumente verwenden wir das gleiche $\mathbf{W}$, und beim Lernen haben wir einen großen Korpus von Dokumenten gegeben, d. h., wir beobachten dann nur die x_i^d-Werte. Wir können dann wie üblich eine a-posteriori-Verteilung aufschreiben und die Wortwahrscheinlichkeiten $\mathbf{W}$ für Themen lernen, die im gesamten Korpus vorkommen.

Nachdem die Matrix $\mathbf{W}$ gelernt ist, entspricht jede ihrer Zeilen einem Thema. Indem wir nach Wörtern mit hoher Wahrscheinlichkeit schauen, können wir diesen Themen so etwas wie Bedeutung zuordnen. Dabei muss aber betont werden, dass wir in jedem Fall eine Matrix $\mathbf{W}$ lernen werden – ob die Zeilen etwas bedeuten oder nicht, ist eine andere Sache.

Das soeben vorgestellte Modell ist parametrisch und seine Größe ist fest. Wir können es nichtparametrisch machen, indem wir K, die Anzahl der Themen, die hier der verborgene Komplexitätsparameter ist, bei Bedarf erhöhen und mithilfe eines Dirichlet-Prozesses an die Daten anpassen. Dabei müssen wir allerdings sorgfältig vorgehen. Jedes Dokument enthält N Wörter, die aus bestimmten Themen stammen; wir haben aber mehrere Dokumente, die alle die gleiche Menge von Themen teilen, d. h., wir müssen die Dirichlet-Prozesse verbinden, die die Themen generieren. Um dies zu erreichen, definieren wir eine Hierarchie. Wir definieren einen übergeordneten Dirichlet-Prozess, aus dem wir die Dirichlet-Prozesse für individuelle Dokumente ziehen. Dies ist ein *hierarchischer Dirichlet-*

HIERARCHISCHER DIRICHLET-PROZESS

Prozess (Teh et al. 2006), der es uns gestattet, Themen, die für ein Dokument gelernt wurden, für alle zu nutzen.

16.12 Betaprotzesse und indische Büffets

Betrachten wir nun ein Beispiel für den Bayesschen Ansatz bei der Dimensionalitätsreduktion im Rahmen der Faktorenanalyse. Gegeben ist in diesem Fall eine $N \times d$-Matrix von Daten $\mathbf{X}$, und wir wollen k Merkmale oder latente Faktoren – Vektoren der Länge d – finden, dies es gestatten, die Daten als Linearkombination von ihnen zu schreiben. Das heißt, wir wollen $\mathbf{Z}$ und $\mathbf{A}$ finden, mit denen wir schreiben können

$$\mathbf{X} = \mathbf{ZA} \,.$$

Dabei ist $\mathbf{A}$ die $k \times d$-Matrix, deren Zeile j der d-dimensionale Merkmalsvektor ist (ähnlich einem Eigenvektor in PCA, siehe Abschnitt 6.3), und $\mathbf{Z}$ ist eine $N \times k$-Matrix, deren Zeile t die Instanz t als einen Vektor von Merkmalen definiert.

Wir nehmen an, dass die z_j^t binär sind und mit der Wahrscheinlichkeit $\boldsymbol{\mu}_j$ aus einer Bernoulli-Verteilung gezogen wurden:

$$z_j^t = \begin{cases} 1 & \text{mit Wahrscheinlichkeit } \mu_j \,, \\ 0 & \text{mit Wahrscheinlichkeit } 1 - \mu_j \,. \end{cases} \tag{16.50}$$

z_j^t zeigt also die Abwesenheit bzw. Anwesenheit des verborgenen Faktors j bei der Konstruktion von Instanz i an. Wenn der entsprechende Faktor vorhanden ist, wird Zeile j von $\mathbf{A}$ gewählt, und die Summe aller so gewählten Zeilen bildet Zeile t von $\mathbf{X}$.

Da wir hier dem Bayesschen Ansatz folgen, definieren wir a-priori-Verteilungen. Wir definieren einen Gaußschen Prior für $\mathbf{A}$ und einen konjugierten Beta-Prior für die μ_j der Bernoullischen z_j^t:

$$\mu_j \sim \text{Beta}(\alpha, 1) \,. \tag{16.51}$$

Dabei ist α der Hyperparameter. Wir können den Posterior aufschreiben und die Matrix $\mathbf{A}$ schätzen. Wenn wir uns die Zeilen von $\mathbf{A}$ anschauen, können wir eine Vorstellung davon bekommen, was die verborgenen Parameter darstellen. Für kleine k (zum Beispiel 2) können wir die Daten skizzieren und visualisieren.

Da wir ein bestimmtes k annehmen, ist dieses Modell parametrisch. Wir können es nichtparametrisch machen und zulassen, dass k infolge weiterer Daten wächst (Griffiths und Ghahramani 2011). Dies definiert einen *Betaprozess* und die dazugehörige Metapher wird *indisches Büffet* genannt. Dieser Prozess definiert ein generatives Modell, das wie folgt funktioniert.

BETAPROZESS
INDISCHES BÜFFET

In einem indischen Restaurant gibt es ein Büffet mit k Speisen, und jeder Gast kann sich von einer Teilmenge dieser Speisen jeweils eine Portion nehmen. Der erste Gast (Instanz 1) kommt herein und nimmt sich von den ersten m Speisen je eine Portion. Wir nehmen an, dass m eine Zufallsvariable ist, die aus einer Poisson-Verteilung mit dem Parameter α generiert wurde. Jeder nachfolgende Gast n nimmt sich mit der Wahrscheinlichkeit n_j/n eine Portion von jeder noch vorhandenen Speise j, wobei n_j die Anzahl der Gäste ist, die sich vor ihm schon eine Portion von j genommen haben. Nachdem der Gast seine Stichprobe an Speisen vom Büffet ausgewählt hat, bittet er außerdem gemäß einer Poisson-Verteilung (α/n) um neue Speisen, wodurch das Modell wächst. Angewendet auf das zuvor diskutierte Faktorenmodell entspricht dies einem Modell, bei dem die Anzahl der Faktoren nicht festgehalten wird, sondern stattdessen mit der inherenten Komplexität der Daten wächst.

16.13 Anmerkungen

Der Bayessche Ansatz hat in den letzten Jahren große Verbreitung gefunden. Die Verwendung von generativen Graphenmodellen deckt sich sehr gut mit dem Bayesschen Formalismus, und es sind interessante Anwendungen in verschiedenen Bereichen zu beobachten, die von der natürlichen Sprachverarbeitung über das maschinelle Sehen bis hin zur Bioinformatik reichen.

Das noch junge Gebiet der nichtparametrischen Bayesschen Modelle ist auch insofern interessant, als die Anpassung der Modellkomplexität nun ein Teil des Trainings ist und nicht in einer äußeren Schleife vorgenommen werden muss. In dieser Richtung ist in naher Zukunft mit weiteren Arbeiten zu rechnen. Ein Beispiel hierfür sind die infiniten Hidden-Markov-Modelle (Beal, Ghahramani und Rasmussen 2002), bei denen die Zahl der verborgenen Zustände automatisch neu eingestellt wird, wenn mehr Daten vorliegen.

Aus Platzgründen wurden die Approximations- und Sampling-Verfahren in diesem Kapitel nicht ausführlicher diskutiert; weitere Informationen zu Variationsverfahren und zum Markov-Ketten-Monte-Carlo-Sampling sind in MacKay 2003, Bishop 2006 sowie Murphy 2012 zu finden.

Der Bayessche Ansatz ist interessant und vielversprechend, und er ist auch bereits in vielen Fällen erfolgreich eingesetzt worden. Dennoch ist er weit davon entfernt, die nicht-Bayessche, frequentistische Sichtweise zu verdrängen. Was die Kontrollierbarkeit betrifft, können generative Modelle zu einfach sein – beispielsweise wird bei der Dirichlet-Analyse die Reihenfolge der Wörter aufgegeben –, es kann schwierig sein, Approximationsverfahren herzuleiten und Sampling-Methoden konvergieren unter Umständen schlecht. Aus diesen Gründen sind frequentistische Abkürzungen (zum Beispiel ein empirischer Bayes-Prior) in bestimmten

Fällen vorzuziehen. Insgesamt empfiehlt es sich, nach dem bestmöglichen Kompromiss zwischen den beiden Welten zu suchen, anstatt sich ausschließlich auf eine Sichtweise festzulegen.

16.14 Übungen

1. Betrachten Sie die Situation in Abbildung 16.3 und beschreiben Sie, wie sich die a-posteriori-Wahrscheinlichkeit ändert, wenn wir N, σ^2 und σ_0^2 ändern.

2. Sei x die Anzahl der Spam-Mails, die in einer zufälligen Stichprobe von n E-Mails enthalten sind. Wir wollen für den Anteil q der Spam-Mails als a-priori-Verteilung eine Gleichverteilung in $[0, 1]$ annehmen. Bestimmen Sie die a-posteriori-Verteilung für $p(q|x)$.

3. Nehmen Sie wie oben an, dass $p(q) \sim \mathcal{N}(\mu_0, \sigma_0^2)$. Nehmen Sie außerdem an, dass n hinreichend groß ist, so dass Sie den zentralen Grenzwertsatz anwenden und die Binomialverteilung durch eine Gauß-Verteilung ersetzen können. Leiten Sie $p(q|x)$ ab.

4. Was ergibt sich für $\mathrm{Var}(r')$, wenn der Maximum-Likelihood-Schätzer verwendet wird? Vergleichen Sie dies mit Gleichung 16.25.

5. Wie ändert sich die Anpassung in Abbildung 16.10, wenn wir s^2 ändern?

   Lösung: Wie üblich ist s der Glättungsparameter, so dass wir glattere Anpassungen bekommen, wenn wir s erhöhen.

6. Schlagen Sie einen Filteralgorithmus vor, um bei Gaußschen Prozessen eine Teilmenge der Trainingsmenge zu wählen.

   Lösung: Eine nützliche Eigenschaft von Gaußschen Prozessen besteht darin, dass wir an jedem beliebigen Punkt die Varianz berechnen können. Für jede Instanz aus der Trainingsmenge können wir dort die Leave-one-out-Schätzung berechnen und prüfen, ob die tatsächliche Ausgabe beispielsweise im 95 %-Vorhersageintervall liegt. Wenn dies der Fall ist, brauchen wir diese Instanz nicht und können sie weglassen. Diejenigen Instanzen, die nicht verworfen werden können, entsprechen den Stützvektoren in einer Kernel-Maschine, also den Instanzen, die gespeichert und benötigt werden, um den Gesamtfehler der Anpassung zu beschränken.

7. Von *aktivem Lernen* spricht man, wenn der Lerner in der Lage ist, x selbst zu generieren und einen Überwacher zu bitten, während des Lernens sukzessive die zugehörigen r-Werte zur Verfügung zu stellen, anstatt passiv eine gegebene Trainingsmenge zu verarbeiten. Wie können wir mithilfe von Gaußschen Prozessen einen solchen

aktives Lernen

aktiven Lerner implementieren? (Hinweis: Wo haben wir die größte Ungewissheit?)

LÖSUNG: Hier verhält es sich ähnlich wie bei der vorherigen Übung, jedoch mit dem Unterschied, dass wir hinzufügen anstatt wegzunehmen. Wenn wir die gleichen Überlegungen wie zuvor anstellen, dann sehen wir, dass wir dort Instanzen brauchen, wo das Vorhersageintervall groß ist. Wir suchen daher nach lokalen Maxima der Varianz, die als Funktion von $\boldsymbol{x}$ gegeben ist. Im Falle eines Gauß-Kernels erwarten wir, dass Punkte, die von den Trainingsdaten entfernt liegen, eine hohe Varianz haben. Aber das muss nicht für alle Kernel gelten. Während der Suche muss sichergestellt sein, dass wir die gültigen Eingabeschranken nicht übertreten.

8. Angenommen, wir haben eine Menge von Dokumenten, und zwar jeweils eine Kopie in Englisch und eine in Französisch. Wie können wir die Methode der latenten Dirichlet-Allokation für diesen Fall erweitern?

16.15 Literaturangaben

Beal, M. J., Z. Ghahramani, and C. E. Rasmussen. 2002. „The Infinite Hidden Markov Model." In *Advances in Neural Information Processing Systems 14,* ed. T. G. Dietterich, S. Becker, and Z. Ghahramani, 577–585. Cambridge, MA: MIT Press.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning.* New York: Springer.

Blei, D. M. 2012. „Probabilistic Topic Models." *Communications of the ACM* 55 (4): 77–84.

Blei, D. M., A. Y. Ng, and M. I. Jordan. 2003. „Latent Dirichlet Allocation." *Journal of Machine Intelligence* 3: 993–1022.

Figueiredo, M. A. T. 2003. „Adaptive Sparseness for Supervised Learning." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25: 1150–1159.

Gershman, S. J., and D. M. Blei. 2012. „A Tutorial on Bayesian Nonparametric Models." *Journal of Mathematical Psychology* 56: 1–12.

Griffiths, T. L., and Z. Ghahramani. 2011. „The Indian Buffet Process: An Introduction and Review." *Journal of Machine Learning Research* 12: 1185–1224.

Hastie, T., R. Tibshirani, and J. Friedman. 2011. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* 2nd ed. New York: Springer.

Hoff, P. D. 2009. *A First Course in Bayesian Statistical Methods.* New York: Springer.

Jordan, M. I., Z. Ghahramani, T. S. Jaakkola, L. K. Saul. 1999. „An Introduction to Variational Methods for Graphical Models.“ *Machine Learning* 37: 183–233.

MacKay, D. J. C. 1998. „Introduction to Gaussian Processes.“ In *Neural Networks and Machine Learning,* ed. C. M. Bishop, 133–166. Berlin: Springer.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms.* Cambridge, UK: Cambridge University Press.

Murphy, K. P. 2007. Conjugate Bayesian Analysis of the Gaussian Distribution. `www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf`.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective.* Cambridge, MA: MIT Press.

Rasmussen, C. E. , and C. K. I. Williams. 2006. *Gaussian Processes for Machine Learning.* Cambridge, MA: MIT Press.

Teh, Y. W., M. I. Jordan, M. J. Beal, and D. M. Blei. 2006. „Hierarchical Dirichlet Processes.“ *Journal of Americal Statistical Association* 101: 1566–1581.

Tibshirani, R. 1996. „Regression Shrinkage and Selection via the Lasso.“ *Journal of the Royal Statistical Society B* 58: 267–288.

Waterhouse, S., D. MacKay, and T. Robinson. 1996. „Bayesian Methods for Mixture of Experts.“ In *Advances in Neural Information Processing Systems 8,* ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 351–357. Cambridge, MA: MIT Press.

17 Kombination mehrerer Lerner

Wir haben in den vergangenen Kapiteln viele verschiedene Lernalgorithmen diskutiert. Zwar sind diese im Allgemeinen erfolgreich, jedoch ist kein einzelner Algorithmus immer der genaueste. Nun werden wir uns mit Modellen befassen, die aus mehreren Lernern bestehen, die einander ergänzen, so dass wir durch die Kombination derer eine höhere Genauigkeit erzielen.

17.1 Grundprinzip

In jeder beliebigen Applikation können wir einen von mehreren Lernalgorithmen nutzen, und bei gewissen Algorithmen gibt es Hyperparameter, welche den finalen Lerner beeinflussen. In einer Klassifikationsaufgabe beispielsweise können wir parametrische Klassifikatoren oder ein mehrlagiges Perzeptron verwenden, und bei einem mehrlagigen Perzeptron sollten wir zum Beispiel bezüglich der Anzahl an verborgenen Einheiten eine Entscheidung treffen. Das Nichts-ist-umsonst-Theorem sagt aus, dass es keinen einzelnen Lernalgorithmus gibt, der in jeder beliebigen Domäne immer den genauesten Lerner erzeugt. Gewöhnlich besteht der Ansatz darin, viele auszuprobieren und denjenigen auszuwählen, der am besten an einem separaten Validierungsdatensatz arbeitet.

Jeder Lernalgorithmus diktiert ein gewisses Modell, welches mit einer Menge an Annahmen einherkommt. Diese induktive Verzerrung führt zu einem Fehler, wenn die Annahmen nicht auf die Daten zutreffen. Lernen ist ein unterbestimmtes Problem, und für einen endlichen Datensatz wird jeder Algorithmus zu einer anderen Lösung gelangen und unter bestimmten Umständen versagen. Die Leistung eines Lerners kann feinabgestimmt werden, um die höchstmögliche Genauigkeit am Validierungsdatensatz zu erzielen, jedoch ist diese Feinabstimmung eine komplexe Aufgabe und es gibt immer noch Instanzen, für die selbst der beste Lerner nicht genau genug ist. Die Idee ist, dass es einen anderen Lerner geben könnte, der an diesen Daten genauer arbeitet. Durch geeignete Kombination von mehreren *Basislernern* kann die Genauigkeit verbessert werden. Mit sinkenden Hardwarekosten wurden solche aus mehreren Lernern bestehende Systeme immer beliebter (Kuncheva 2004).

Basislerner

Es gibt hierbei zwei grundsätzliche Fragen:

1. Wie können wir Basislerner generieren, die einander ergänzen?
2. Wie kombinieren wir die Ausgaben der Basislerner, um maximale Genauigkeit zu erreichen?

Die Diskussion in diesem Kapitel wird diese beiden miteinander zusammenhängenden Fragen beantworten. Wir werden feststellen, dass die Kombination von Modellen kein Trick ist, der die Genauigkeit in jedem Fall erhöht. Sie führt dagegen immer zum Anwachsen der zeitlichen und räumlichen Komplexität des Trainings und des Testens, und wenn wir die Basislerner nicht sorgfältig trainieren und ihre Entscheidungen nicht klug kombinieren, werden wir einfach nur für diese zusätzliche Komplexität bezahlen, ohne dafür einen signifikanten Gewinn an Genauigkeit zu bekommen.

17.2 Generierung diverser Lerner

Diversität

Da es keinen Sinn hätte, Lerner zu kombinieren, die immer ähnliche Entscheidungen treffen, besteht das Ziel darin, eine Menge von *diversen Lernern* zu finden, die sich in ihren Entscheidungen unterscheiden, und zwar so, dass sie einander ergänzen. Gleichzeitig kann es keinen Gewinn im Sinne eines höheren Gesamterfolgs geben, wenn die Lerner nicht wenigstens in ihrem Kompetenzbereich genau sind. Wir stehen daher vor der doppelten Aufgabe, sowohl die individuellen Genauigkeiten als auch die Diversität zwischen den Lernern zu maximieren. Wir wollen nun verschiedene Möglichkeiten diskutieren, wie wir das erreichen können.

Verschiedene Algorithmen

Wir können verschiedene Lernalgorithmen nutzen, um die unterschiedlichen Basislerner zu trainieren. Verschiedene Algorithmen treffen unterschiedliche Annahmen hinsichtlich der Daten und führen zu verschiedenen Klassifikatoren. Zum Beispiel mag der eine Basislerner parametrisch sein und ein anderer nichtparametrisch. Wenn wir uns für einen einzelnen Algorithmus entscheiden, betonen wir eine einzelne Methode und ignorieren alle anderen. Die Kombination von mehreren Lernern, die auf mehreren Algorithmen basieren, befreit uns von der Aufgabe der Entscheidungsfindung und wir müssen nicht mehr alles auf eine Karte setzen.

Verschiedene Hyperparameter

Wir können denselben Lernalgorithmus nutzen, aber dabei unterschiedliche Hyperparametern verwenden. Beispiele sind die Anzahl an verbor-

genen Einheiten in einem mehrlagigen Perzeptron, k bei der k-Nächste-Nachbarn-Methode, der Fehlerschwellwert in Entscheidungsbäumen, und so weiter. Bei einem Gaußschen parametrischen Klassifikator – egal ob die gleichen Kovarianzmatrizen für die verschiedenen Klassen genutzt werden oder nicht – handelt es sich um einen Hyperparameter. Wenn der Optimierungsalgorithmus eine iterative Prozedur verwendet wie den Gradientenabstieg, dessen finaler Zustand vom Ausgangszustand abhängt, wie bei der Rückpropagierung mit mehrlagigen Perzeptronen, dann ist der Ausgangszustand, also zum Beispiel die Ausgangsgewichtungen, ein weiterer Hyperparameter. Wenn wir multiple Basislerner mit verschiedenen Hyperparameterwerten trainieren, dann bilden wir darüber den Durchschnitt und reduzieren die Varianz und somit auch den Fehler.

Verschiedene Repräsentationen der Eingabe

Separate Basislerner können auch unterschiedliche *Repräsentationen* vom selben Eingabeobjekt oder -ereignis nutzen, wodurch die Integration verschiedener Typen von Sensoren/Maßeinheiten oder Merkmalen ermöglicht wird. Verschiedene Repräsentationen bringen unterschiedliche Charakteristika explizit zum Vorschein, wodurch bessere Identifikationen möglich werden. In vielen Anwendungen gibt es multiple Informationsquellen, und es ist wünschenswert, all diese Daten zu nutzen, um mehr Informationen zu extrahieren und höhere Genauigkeit bei Vorhersagen zu erzielen.

Um beispielsweise bei der Spracherkennung die geäußerten Wörter zu erkennen, können wir zusätzlich zur akustischen Eingabe auch das Videobild der Lippen des Sprechers zur Hilfe nehmen, wenn die Wörter ausgesprochen werden. Dies ähnelt der *Sensordatenfusion*, bei der Daten aus verschiedenen Sensoren integriert werden, um mehr Informationen für eine spezifische Anwendung zu extrahieren. Ein anderes Beispiel ist das Information Retrieval, etwa für Bilddaten, die zusätzlich zu dem Bild selbst Textdaten in Form von Keywords umfassen. In diesem Fall wollen wir in der Lage sein, die Informationen aus beiden Quellen zu kombinieren, um die richtige Menge von Bildern zu finden. Dies wird auch *Mehrfachlernen* genannt.

Sensordatenfusion

Mehrfachlernen

Der einfachste Weg ist der, alle Datenvektoren zu verknüpfen und sie wie einen großen Vektor aus einer einzelnen Quelle zu behandeln, doch scheint dies theoretisch unangebracht, da es der Modellierung von Daten gemäß einer Stichprobenerhebung aus einer multivariaten statistischen Verteilung gleichkommt. Weiterhin gestalten große Eingabedimensionalitäten das System komplexer und bedingen größere Stichproben, damit die Schätzer genau sind. Der von uns gewählte Ansatz ist der, separate Entscheidungen, basierend auf verschiedenen Quellen, zu treffen und unterschiedliche Basislerner zu verwenden, bevor dann ihre Vorhersagen kombiniert werden.

Selbst wenn es nur eine einzige Repräsentation der Eingabe gibt, können wir Klassifikatoren haben, die verschiedene Eingabemerkmale nutzen, wenn wir zufällige Teilmengen der Eingabe auswählen. Dieser Ansatz wird *Methode des zufälligen Teilraums* genannt (Ho 1998). Wir erreichen dadurch Robustheit, weil verschiedene Lerner das gleiche Problem aus verschiedenen Perspektiven betrachten. Außerdem reduzieren wir so das Problem der Dimensionalität, da die Eingaben weniger Dimensionen haben.

ZUFÄLLIGER TEILRAUM

Verschiedene Trainingsmengen

Eine weitere Möglichkeit bietet die Verwendung von *verschiedenen Trainingsmengen*, um die unterschiedlichen Basislerner zu trainieren. Dies kann nach dem Zufallsprinzip geschehen, indem zufällige Trainingsdaten aus der gegebenen Stichprobe gezogen werden; hierbei spricht man vom *Bagging-Algorithmus*. Die Lerner können aber auch seriell trainiert werden, so dass Instanzen, für welche die vorhergehenden Basislerner nicht akkurat sind, mehr Betonung beim Training späterer Basislerner erhalten; Beispiele sind der *Boosting-Algorithmus* und das *Kaskadieren*, wobei aktiv versucht wird, komplementäre Lerner zu generieren, statt dies dem Zufall zu überlassen.

Die Partitionierung der Trainingsstichprobe kann ebenfalls basierend auf Lokalität im Eingaberaum vorgenommen werden, so dass jeder Basislerner an Instanzen in einer gewissen lokalen Region des Eingaberaums trainiert wird; genau das wird durch *gemischte Expertensysteme* erreicht, die wir in Kapitel 12 diskutiert haben, die wir hier aber erneut in diesem Zusammenhang der Kombination von mehreren Lernern betrachten werden. Auf ähnliche Weise ist es möglich, die Hauptaufgabe hinsichtlich einer Zahl an durch die Basislerner zu implementierenden Unteraufgaben zu definieren, wie dies durch *Fehlerkorrekturcodes* erreicht wird.

Diversität vs. Genauigkeit

Eine wichtige Bemerkung ist die, dass wir bei der Generierung von multiplen Basislernern wollen, dass diese zu einem vernünftigen Maß genau sind, von ihnen aber nicht verlangen, individuell sehr akkurat sein zu müssen; somit sind sie nicht separat für die bestmögliche Genauigkeit optimiert und müssen dies auch nicht sein. Die Basislerner werden nicht wegen ihrer Genauigkeit gewählt, sondern wegen ihrer Einfachheit. Wir fordern jedoch, dass die Basislerner an unterschiedlichen Instanzen genau arbeiten und sich in Subdomänen des Problems spezialisieren. Was uns wichtig ist, ist nicht die Genauigkeit der Basislerner, mit denen wir begonnen hatten, sondern die finale Genauigkeit, wenn die Basislerner kombiniert werden. Nehmen wir an, wir haben einen Klassifikator, der 80-prozentig genau ist. Wenn wir uns bezüglich eines zweiten Klassifikators entscheiden, kümmert uns nicht die insgesamte Genauigkeit; wir

interessieren uns nur für die Genauigkeit bei den 20 Prozent, welche vom ersten Klassifikator fehlklassifiziert werden, so lange wir wissen, wann welcher der beiden zu verwenden ist.

Das bedeutet, dass die Forderungen an die Genauigkeit und die Diversität der Lerner auch davon abhängen, wie ihre Entscheidungen kombiniert werden. Dies wollen wir im Anschluss diskutieren. Wenn, wie im Falle eines Voting-Schemas, ein Lerner für alle Eingaben befragt wird, dann sollte er überall genau sein und Diversität sollte überall erzwungen werden. Wenn wir dagegen eine Partitionierung des Eingaberaums in Kompetenzbereiche für die verschiedenen Lerner haben, ist die Diversität durch die Partitionierung bereits garantiert und die Lerner müssen nur in ihren jeweiligen Bereich genau sein.

17.3 Methoden der Modellkombination

Zusätzlich zur Art und Weise, wie die Lerner trainiert werden, existieren auch verschiedene Arten, die multiplen Basislerner zu kombinieren, um die finale Ausgabe zu generieren:

- *Methoden zur Kombination mehrerer Expertensysteme* haben Basislerner, die *parallel* arbeiten. Diese Methoden können ihrerseits in zwei Gruppen unterteilt werden: KOMBINATION VON EXPERTENSYSTEMEN
  - Beim *globalen Ansatz,* der auch als *Fusion von Lernern* bezeichnet wird, generieren alle Basislerner für eine gegebene Eingabe eine Ausgabe, und alle diese Ausgaben werden verwendet. Beispiele für diesen Ansatz sind das *Voting* und die *Schachtelung.*
  - Beim *lokalen Ansatz,* der auch *Selektion von Lernern* genannt wird und beispielsweise in *gemischten Expertensystemen* angewendet wird, gibt es ein *Gattermodell,* das die Eingabe betrachtet und einen (oder einige wenige) Lerner auswählt, der für das Generieren der Ausgabe herangezogen wird.
- *Mehrstufige Kombinationsmethoden* nutzen einen *seriellen* Ansatz, bei dem der nächste Basislerner nur mit den Instanzen trainiert oder getestet wird, bei denen die vorhergehenden Basislerner nicht akkurat genug waren. Die Idee besteht darin, die Basislerner (oder die verschiedenen von ihnen genutzten Repräsentationen) nach zunehmender Komplexität zu sortieren, so dass ein komplexer Basislerner nicht verwendet wird (oder seine komplexe Repräsentation nicht extrahiert wird), es sei denn, die vorhergehenden einfacheren Basislerner sind nicht konfident genug. Ein Beispiel bildet das Verfahren der *Kaskadierung.* MEHRSTUFIGE KOMBINATIONSMETHODEN

Nehmen wir an, wir haben L Basislerner. Wir bezeichnen die Vorhersage eines Basislerners $\mathcal{M}_j$ bei gegebener willkürlicher dimensionaler Eingabe x mit $d_j(x)$. Im Falle von mehreren Repräsentationen nutzt jedes $\mathcal{M}_j$ eine unterschiedliche Eingaberepräsentation x_j. Die finale Vorhersage wird anhand der Vorhersagen der Basislerner errechnet:

$$y = f(d_1, d_2, \ldots, d_L|\Phi) \tag{17.1}$$

mit $f(\cdot)$ als Kombinationsfunktion mit den Parametern Φ. Wenn es K Ausgaben gibt, dann hat jeder Lerner K Ausgaben, $d_{ji}(x), i = 1, \ldots, K$, $j = 1, \ldots, L$, und durch deren Kombination erzeugen wir auch K Werte, $y_i, i = 1, \ldots, K$; dann wählen wir beispielsweise in der Klassifikation die Klasse mit dem maximalen y_i-Wert.

17.4 Voting

VOTING

Am einfachsten kombiniert man mehrere Klassifikatoren durch *Voting* (Abstimmen), was dem Bilden einer Linearkombination der Lerner entspricht. (siehe Abbildung 17.1)

$$y = \sum_{j=1}^{L} w_j d_j \;\text{ mit }\; w_j \geq 0, \forall j \text{ und } \sum_{j=1}^{L} w_j = 1\,. \tag{17.2}$$

ENSEMBLE
LINEARER
MEINUNGSPOOL

Diese sind auch als *Ensembles* oder *lineare Meinungspools* bekannt. Im einfachsten Fall haben alle Lerner das gleiche Gewicht und wir haben ein *einfaches Voting,* was der Bildung des Mittelwerts entspricht. Allerdings ist das Bilden einer (gewichteten) Summe nur eine von mehreren

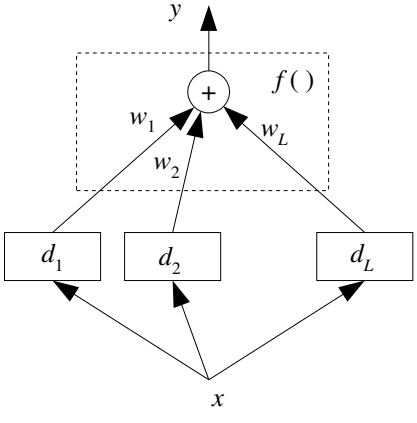


Abb. 17.1: Beim Voting ist die Kombinationsfunktion $f(\cdot)$ eine gewichtete Summe. d_j sind die mehreren Lerner und w_j sind die Gewichte ihrer Stimmabgaben. y ist die insgesamte Ausgabe. Im Falle von multiplen Ausgaben, beispielsweise bei der Klassifikation, haben die Lerner multiple Ausgaben d_{ji}, deren gewichtete Summe y_i liefert. Man beachte außerdem, dass im Diagramm alle Lerner dieselbe Eingabe beobachten; es kann der Fall eintreten, dass verschiedene Lerner unterschiedliche Repräsentationen desselben Eingabeobjektes oder -ereignisses beobachten.

Möglichkeiten. Weitere Kombinationsregeln sind in Tabelle 17.1 zusammengestellt (Kittler et al. 1998). Wenn die Ausgaben keine a-posteriori-Wahrscheinlichkeiten sind, verlangen diese Regeln, dass die Ausgaben auf die gleiche Skala normiert werden müssen (Jain, Nandakumar und Ross 2005).

Tabelle 17.1: Kombinationsregeln für Klassifikatoren

Regel	Fusionsfunktion
Summe	$y_i = \frac{1}{2}\sum_{j=1}^{L} d_{ji}$
gewichtete Summe	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max d_{ji}$
Produkt	$y_i = \prod_j d_{ji}$

Tabelle 17.2 zeigt ein Beispiel für die Anwendung dieser Regeln, wobei die Effekte der verschiedenen Regeln deutlich werden. Die Summenregel ist am intuitivsten und sie ist zugleich die in der Praxis am häufigsten eingesetzte. Die Medianregel ist robuster gegenüber Ausreißern; die Minimum- und die Maximumregel sind pessimistisch bzw. optimistisch. Bei der Produktregel hat jeder Lerner ein Vetorecht, d. h., wenn einer der Lerner eine Ausgabe von 0 hat, wird die Gesamtausgabe unabhängig von allen anderen Ausgaben ebenfalls 0. Man beachte, dass sich die y_i nach den Kombinationsregeln nicht notwendig zu 1 addieren.

Tabelle 17.2: Beispiel für Kombinationsregeln mit drei Lernern und drei Klassen

	C_1	C_2	C_2
d_1	0,2	0,5	0,3
d_2	0,0	0,6	0,4
d_3	0,4	0,4	0,2
Summe	0,2	**0,5**	0,3
Median	0,2	**0,5**	0,4
Minimum	0,0	**0,4**	0,2
Maximum	0,4	**0,6**	0,4
Produkt	0,0	**0,12**	0,032

Bei der gewichteten Summe ist d_{ji} das Votum von Lerner j für Klasse C_i und w_j ist das Gewicht dieses Votums. Das einfache Voting ist ein Spezialfall, bei dem alle Wähler das gleiche Gewicht besitzen, nämlich

$w_j = 1/L$. Bei der Klassifikation spricht man hierbei von der *Pluralitätswahl*, wobei die Klasse mit der maximalen Anzahl an Stimmen zum Gewinner ernannt wird. Bei zwei Klassen handelt es sich um die *Mehrheitswahl*, bei der die Gewinnerklasse mehr als die Hälfte aller Stimmen erzielt (Übung 1). Wenn die Wähler außerdem die Möglichkeit haben, die zusätzliche Information bereitzustellen, wie stark sie für jede Klasse votieren (z.B. durch die a-posteriori-Wahrscheinlichkeit), dann können diese Informationen nach der Normalisierung als Gewichte in einem *gewichteten Wahlverfahren* verwendet werden. Ein äquivalenter Fall liegt vor, wenn die d_{ji} die a-posteriori-Wahrscheinlichkeiten der Klassen $P(\mathcal{C}_i|x, \mathcal{M}_j)$ repräsentieren; dann können wir sie einfach summieren ($w_j = 1/L$) und die Klasse mit dem maximalen y_i wählen.

Im Falle der Regression kann ein einfaches oder ein gewichtetes Mittel oder der Median verwendet werden, um die Ausgaben der Basisregressoren zu fusionieren. Der Median ist robuster gegenüber Rauschen als der Mittelwert.

Eine weitere Möglichkeit, die w_j zu finden, besteht darin, die Genauigkeiten der Lerner (Regressor oder Klassifikator) an einem separaten Validierungsdatensatz zu bewerten und diese Information zur Berechnung der Gewichte zu verwenden, so dass wir den genauer arbeitenden Lernern mehr Gewicht zukommen lassen. Diese Gewichte können auch aus den Daten gelernt werden, was wir im Zusammenhang mit der geschachtelten Generalisierung (Abschnitt 17.9) diskutieren werden.

Wahlmethoden können als Approximationen mit Bayesschen Rahmenbedingungen gesehen werden, bei denen Gewichte die a-priori-Wahrscheinlichkeiten des Modells approximieren und Modellentscheidungen modellbezogene Likelihood-Werte approximieren. Dies wird als *Bayessche Modellkombination* bezeichnet (siehe Abschnitt 16.6). In der Klassifikation haben wir zum Beispiel $w_j \equiv P(\mathcal{M}_j)$, $d_{ji} = P(\mathcal{C}_i|x, \mathcal{M}_j)$, und Gleichung 17.2 entspricht

Bayessche Modellkombination

$$P(\mathcal{C}_i|x) = \sum_{\text{alle Modelle } \mathcal{M}_j} P(\mathcal{C}_i|x, \mathcal{M}_j)P(\mathcal{M}_j) \,. \tag{17.3}$$

Das einfache Voting entspricht einer einheitlichen a-priori-Wahrscheinlichkeit. Wenn wir eine a-priori-Verteilung vorliegen haben, welche einfachere Modelle bevorzugt, dann würden diesen stärkere Gewichte zugewiesen. Wir können nicht über alle Modelle hinweg integrieren; wir sind lediglich in der Lage, eine Teilmenge zu wählen, für die wir glauben, dass $P(\mathcal{M}_j)$ hoch ist; oder wir fügen einen weiteren Bayesschen Schritt ein und berechnen $P(\mathcal{M}_j|\mathcal{X})$, die Wahrscheinlichkeit eines Modells bei vorliegender Stichprobe, und entnehmen eine Stichprobe von Modellen mit hoher Wahrscheinlichkeit aus dieser Dichte.

Hansen und Salamon (1990) haben bewiesen, dass sich bei unabhängigen Klassifikatoren für zwei Klassen mit einer Erfolgswahrscheinlichkeit größer

als 1/2, also besser als zufälliges Raten, die Genauigkeit unter Verwendung des Mehrheitswählens erhöht, wenn sich die Zahl an abstimmenden Klassifikatoren erhöht.

Nehmen wir an, d_j seien unabhängig und identisch verteilt mit Erwartungswert $E[d_j]$ und Varianz $\text{Var}(d_j)$; bilden wir dann einen einfachen Durchschnitt mit $w_j = 1/L$, so ergeben sich Erwartungswert und Varianz der Ausgabe als

$$\begin{aligned} E[y] &= E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L E[d_j] = E[d_j]\,, \\ \text{Var}(y) &= \text{Var}\left(\sum_j \frac{1}{L} d_j\right) = \frac{1}{L^2}\text{Var}\left(\sum_j d_j\right) \\ &= \frac{1}{L^2} L \text{Var}(d_j) = \frac{1}{L}\text{Var}(d_j)\,. \end{aligned} \tag{17.4}$$

Wir erkennen, dass sich der Erwartungswert nicht verändert und somit die Verzerrung ebenfalls unverändert bleibt. Die Varianz jedoch, und damit der mittlere quadratische Fehler, verringert sich mit zunehmender Zahl an unabhängigen Wählern L. Im allgemeinen Fall gilt

$$\begin{aligned} \text{Var}(y) &= \frac{1}{L^2}\text{Var}\left(\sum_j d_j\right) \\ &= \frac{1}{L^2}\left[\sum_j \text{Var}(d_j) + 2\sum_j \sum_{i<j} \text{Cov}(d_j, d_i)\right] \end{aligned} \tag{17.5}$$

und falls die Lerner positiv korreliert sind, bedeutet dies, dass die Varianz (und der Fehler) wächst. Wir können daher die Verwendung unterschiedlicher Algorithmen und Eingabemerkmale als Bemühung ansehen, die positive Korrelation zu verringern oder gar vollständig zu eliminieren.

Wir sehen außerdem, dass eine weitere Abnahme der Varianz möglich ist, wenn die Wähler nicht unabhängig, aber negativ korreliert sind. Der Fehler nimmt dann ab, wenn die damit einhergehende Zunahme der Verzerrung nicht höher ist, weil diese Ziele sich widersprechen. Wir können keine große Zahl von Klassifikatoren haben, die alle genau *und* dabei negativ korreliert sind. Bei gemischten Expertensystemen zum Beispiel, wo die Lerner lokalisiert sind, sind die Experten negativ korreliert aber verzerrt (Jacobs 1997).

Wenn wir jeden Basislerner als eine Funktion, die zufälliges Rauschen der wahren Diskriminanz- oder Regressionsfunkion hinzufügt, auffassen,

dann ist das Mitteln der einzelnen Schätzungen gleich dem Mitteln über das Rauschen, wenn diese Funktionen unkorreliert mit Mittelwert 0 sind. In diesem Sinne hat das Voting den Effekt der Glättung im funktionalen Raum und kann als ein an der wahren Funktion eingesetzter Regulierer mit einer Glättungsvermutung angesehen werden (Perrone 1993). Wir haben dafür bereits in Abbildung 4.5d ein Beispiel betrachtet, bei dem die Durchschnittsbildung über Modelle mit großer Varianz uns eine bessere Anpassung lieferte als mit den individuellen Modellen. Genau das ist die Grundidee des Voting: wir wählen aus Modellen mit hoher Varianz und niedriger Verzerrung, so dass nach der Kombination die Verzerrung gering bleibt und wir die Varianz durch Durchschnittsbildung reduzieren. Selbst, wenn die individuellen Modelle verzerrt sind, wird die Abnahme der Varianz unter Umständen die Verzerrung ausgleichen und eine Fehlerabnahme ist immer noch möglich.

17.5 Fehlerkorrekturcodes

Fehlerkorrekturcodes

Bei *Fehlerkorrekturcodes* (engl. *error-correcting output codes*, ECOC) (Dietterich und Bakiri 1995) ist die hauptsächliche Klassifikationsaufgabe hinsichtlich einer Zahl an Unteraufgaben definiert, welche durch die Basislerner implementiert werden. Der Hintergedanke ist der, dass die ursprüngliche Aufgabe, eine Klasse von allen anderen Klassen zu trennen, sich als schwieriges Problem herausstellen könnte. Stattdessen wollen wir eine Menge an einfacheren Klassifikationsproblemen definieren, von denen jedes auf einen Aspekt der Aufgabe hin spezialisiert ist; indem wir diese einfacheren Klassifikatoren dann kombinieren, erhalten wir den finalen Klassifikator.

Basislerner sind binäre Klassifikatoren mit den Ausgaben $-1/+1$, und es existiert eine *Codematrix* $\mathbf{W}$ aus $K \times L$, deren K Zeilen den binären Codes der Klassen hinsichtlich der L Basislerner d_j entsprechen. Ist beispielsweise die zweite Zeile von $\mathbf{W}$ gleich $[-1, +1, +1, -1]$, dann bedeutet dies für uns, dass wir von einer Instanz als zu $\mathcal{C}_2$ zugehörig sprechen können, wenn diese Instanz sich auf der negativen Seite von d_1 und d_4 und auf der positiven Seite von d_2 und d_3 befindet. Auf ähnliche Weise definieren die Spalten der Codematrix die Aufgaben der Basislerner. Wenn zum Beispiel die dritte Spalte $[-1, +1, +1]^T$ lautet, dann lesen wir daran ab, dass die Aufgabe des dritten Basislerners, d_3, darin besteht, die Instanzen von $\mathcal{C}_1$ von den kombinierten Instanzen von $\mathcal{C}_2$ und $\mathcal{C}_3$ zu trennen. Auf diese Weise formen wir die Trainingsmenge für die Basislerner. In diesem Fall beispielsweise wird $\mathcal{X}_3^+$ aus allen Instanzen mit Zuordnung $\mathcal{C}_2$ und $\mathcal{C}_3$ gebildet, Instanzen mit Zuordnung $\mathcal{C}_1$ formen $\mathcal{X}_3^-$, und d_3 wird so trainiert, dass $x^t \in \mathcal{X}_3^+$ die Ausgabe $+1$ und $x^t \in \mathcal{X}_3^-$ die Ausgabe -1 liefert.

Die Codematrix erlaubt es uns somit, eine Entscheidung für eine aus mehreren Klassen (Klassifikationsproblem mit $K > 2$) mittels Dichoto-

mien (Klassifikationsprobleme mit $K = 2$) zu definieren und entspricht einer Methode, die unter Nutzung beliebiger Lernalgorithmen zur Implementierung der dichotomischen Basislerner angewandt werden kann – zum Beispiel mit linearen oder mehrlagigen Perzeptronen (mit einzelner Ausgabe), Entscheidungsbäumen oder einer SVM, deren ursprüngliche Definition für Zweiklassenprobleme ausgelegt ist.

Die typische Fall eines Diskriminators pro Klasse entspricht der diagonalen Codematrix mit $L = K$. Für $K = 4$ erhalten wir zum Beispiel

$$\mathbf{W} = \begin{bmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{bmatrix}.$$

Das Problem hier ist, dass im Falle eines Fehlers bei einem der Basislerner eine Fehlklassifikation verursacht wird, da die Klassencodewörter so ähnlich sind. Daher besteht der Ansatz der Fehlerkorrekturcodes darin, mit $L > K$ zu arbeiten und die Hamming-Distanz zwischen den Codewörtern zu erhöhen. Eine Möglichkeit bietet die *paarweise Trennung* von Klassen, bei der es einen separaten Basislerner gibt, um $\mathcal{C}_i$ von $\mathcal{C}_j$ zu trennen, für $i < j$ (Abschnitt 10.4). In diesem Fall ist $L = K(K-1)/2$, und mit $K = 4$ erhalten wir die Codematrix

$$\mathbf{W} = \begin{bmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix},$$

bei der ein Nulleintrag für „egal“ steht. Das heißt, d_1 wird trainiert, um $\mathcal{C}_1$ von $\mathcal{C}_2$ zu trennen und nutzt keine der Trainingsinstanzen, die zu den anderen Klassen gehören. Auf vergleichbare Weise stellen wir fest, dass eine Instanz zu $\mathcal{C}_2$ gehört, wenn $d_1 = -1$ und $d_4 = d_5 = +1$, und wir ignorieren die Werte von d_2, d_3, und d_6. Das Problem dabei ist, dass L gleich $\mathcal{O}(K^2)$ ist und für große K die paarweise Trennung möglicherweise nicht durchführbar ist.

Wenn wir L groß wählen können, wird es gut funktionieren, die Codematrix einfach zufällig zu generieren, d. h. mit -1 und $+1$ zu füllen, aber wenn wir L klein halten wollen, müssen wir $\mathbf{W}$ optimieren. Der Ansatz besteht darin, L vorher festzulegen und dann $\mathbf{W}$ so zu bestimmen, dass die Entfernungen zwischen Zeilen und gleichzeitig die Entfernungen zwischen Spalten bezüglich der Hamming-Distanz so groß wie möglich sind. Bei K Klassen gibt es $2^{(K-1)} - 1$ mögliche Spalten, also Zweiklassenprobleme. Das ist dadurch begründet, dass K Bits auf 2^K verschiedene Arten und in Komplementen (z. B. definieren $[-1+1-1+1]^T$ und $[+1-1+1-1]^T$ aus unserer Perspektive dieselbe Diskriminante) geschrieben werden können, wobei die möglichen Kombinationen durch 2 geteilt werden und dann 1

abgezogen wird, da eine Spalte nur aus Nullen (oder Einsen) nutzlos ist. Bei $K = 4$ erhalten wir beispielsweise

$$\mathbf{W} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 \end{bmatrix} .$$

Ist K groß, so suchen wir bei einem gegebenen Wert für L nach L Spalten aus den $2^{(K-1)} - 1$. Wir wollen, dass diese Spalten aus $\mathbf{W}$ so verschieden wie möglich sind, so dass die durch die Basislerner zu erfassenden Aufgaben sich so stark wie möglich voneinander unterscheiden. Gleichzeitig wollen wir auch, dass die Zeilen von $\mathbf{W}$ möglichst unterschiedlich sind, so dass wir die maximale Fehlerkorrektur vornehmen können, falls einer oder mehrere Basislerner versagen.

Die ECOC können als Methode des Voting formuliert werden, wobei die Einträge von $\mathbf{W}$, w_{ij}, als Stimmgewichtungen erachtet werden:

$$y_i = \sum_{j=1}^{L} w_{ij} d_j \, . \tag{17.6}$$

Sodann wählen wir die Klasse mit dem höchsten y_i. Die Verwendung einer gewichteten Summe und die darauffolgende Wahl des Maximums statt der Überprüfung hinsichtlich einer exakten Übereinstimmung ermöglichen es, dass d_j nicht länger binär sein muss, sondern einen Wert zwischen -1 und $+1$ annehmen kann und somit weiche Gewissheiten statt harter Entscheidungen unterstützt. Man beachte, dass ein Wert p_j zwischen 0 und 1, zum Beispiel eine a-posteriori-Wahrscheinlichkeit, in einen Wert d_j zwischen -1 und $+1$ konvertiert werden kann, und zwar einfach durch

$$d_j = 2p_j - 1 \, .$$

Der Unterschied zwischen Gleichung 17.6 und dem generischen Wahlmodell aus Gleichung 17.2 ist der, dass die Gewichte von Stimmabgaben für verschiedene Klassen unterschiedlich sein können, sprich, wir nutzen nicht länger w_j sondern w_{ij}; außerdem ist $w_j \geq 0$, wohingegen die w_{ij} gleich -1, 0 oder $+1$ sind.

Ein Problem mit ECOC ergibt sich, weil wir a priori die Codematrix $\mathbf{W}$ festlegen: dadurch gibt es keine Garantie, dass die durch die Spalten von $\mathbf{W}$ definierten Unteraufgaben einfach sein werden. Dietterich und Bakiri (1995) berichten, dass die Bäume für Entscheidungen für eine aus zwei Klassen größer sein können als die Bäume für Entscheidungen für eine aus mehr als zwei Klassen, und bei der Verwendung von mehrlagigen Perzeptronen kann es zu langsamerer Konvergenz durch Rückpropagierung kommen.

17.6 Bagging

BAGGING

Beim *Bagging* handelt es sich um eine Methode des Voting, bei der Basislerner dadurch unterschiedlich gehalten werden, dass sie mittels leicht unterschiedlicher Trainingsdatensätze trainiert werden. Die Erstellung von L leicht unterschiedlichen Stichproben aus einer gegebenen Stichprobe wird durch den Bootstrap-Algorithmus vorgenommen, wobei wir bei vorliegendem Trainingsdatensatz $\mathcal{X}$ der Größe N zufällig N Instanzen aus $\mathcal{X}$ *mit Zurücklegen* ziehen. Da die Stichprobenerhebung mit Zurücklegen erfolgt, ist es möglich, dass einige Instanzen mehr als einmal und andere Instanzen überhaupt nicht gezogen werden. Wird dies vollzogen, um L Stichproben $\mathcal{X}_j, j = 1, \ldots, L$ zu generieren, dann ähneln sich diese Stichproben, da sie alle derselben ursprünglichen Stichprobe entstammen, jedoch unterscheiden sie sich auch ein klein wenig aufgrund des zufälligen Ziehens. Die Basislerner d_j werden mit diesen L Stichproben $\mathcal{X}_j$ trainiert.

INSTABILER ALGORITHMUS

Ein Lernalgorithmus ist ein *instabiler Algorithmus*, wenn kleine Veränderungen im Trainingsdatensatz einen großen Unterschied beim generierten Lerner bewirken, sprich, wenn der Lernalgorithmus hohe Varianz aufweist. Bagging, die Kurzform für *Bootstrap Aggregating*, nutzt den Bootstrap-Algorithmus, um L Trainingsdatensätze zu generieren, trainiert L Basislerner mit Hilfe einer instabilen Lernprozedur und verwendet dann während der Testphase einen Durchschnittswert (Breiman 1996). Bagging kann sowohl für die Klassifikation als auch Regression verwendet werden. Im Falle der Regression kann man aus Gründen stärkerer Robustheit den Median statt des Durchschnitts bei der Kombination von Vorhersagen nutzen.

Wie wir zuvor gesehen haben, reduziert das Mitteln die Varianz nur dann, wenn die positive Korrelation klein ist; ein Algorithmus ist stabil, wenn mehrere Durchläufe des gleichen Algorithmus auf neu erhobenen Versionen der gleichen Datenmenge zu Lernern mit stark positiver Korrelation führen. Algorithmen wie Entscheidungsbäume und mehrlagige Perzeptronen sind instabil. Der Nächste-Nachbarn-Algorithmus ist stabil, der verdichtete Nächste-Nachbarn-Algorithmus ist es aber nicht (Alpaydın 1997). Wenn der ursprüngliche Trainingsdatensatz groß ist, dann ist es unter Umständen ratsam, daraus kleinere Mengen der Größe $N' < N$ mittels der Bootstrap-Methode zu generieren, da andernfalls die Bootstrap-Replikate $\mathcal{X}_j$ zu ähnlich wären und d_j hochkorreliert vorliegen würden.

17.7 Boosting

Beim Bagging wird die Generierung von komplementären Basislernern dem Zufall und der Instabilität der Lernmethode überlassen. Beim Boosting versuchen wir aktiv, komplementäre Basislerner zu generieren, indem

BOOSTING

wir den nachfolgenden Lerner anhand der Fehler der vorhergehenden trainieren. Der ursprüngliche *Boosting-Algorithmus* (Schapire 1990) kombiniert drei schwache Lerner, um einen starken Lerner zu generieren.

SCHWACHER LERNER

Ein *schwacher Lerner* hat eine Fehlerwahrscheinlichkeit kleiner $1/2$, wodurch er bei einem Zweiklassenproblem bessere Ergebnisse erzielt als durch zufälliges Raten, und ein *starker Lerner* hat eine willkürlich kleine Fehlerwahrscheinlichkeit.

STARKER LERNER

Einen gegebenen großen Trainingsdatensatz zerlegen wir zufällig in drei Teile. Wir nutzen $\mathcal{X}_1$ und trainieren d_1. Dann nehmen wir $\mathcal{X}_2$ und klassiffizieren alle Instanzen mittels d_1. Wir nutzen aus $\mathcal{X}_2$ alle durch d_1 fehlklassifizierten Instanzen sowie genauso viele Instanzen, bei denen d_1 richtig lag, und diese zusammen bilden den Trainingsdatensatz von d_2. Daraufhin ziehen wir $\mathcal{X}_3$ und klassiffizieren alle Instanzen mittels d_1 und d_2. Die Instanzen, bei denen d_1 und d_2 uneins sind, bilden den Trainingsdatensatz von d_3. Während der Testphase klassifizieren wir eine gegebene Instanz mit d_1 und d_2; stimmen die Ergebnisse überein, dann ist dies die Antwort; andernfalls wird die Antwort von d_3 als Ausgabe gewählt. Schapire (1990) hat bewiesen, dass dieses umfassende System eine reduzierte Fehlerrate aufweist und dass die Fehlerrate willkürlich reduziert werden kann, wenn so ein System rekursiv verwendet wird, das heißt, ein Boosting-System aus drei Modellen, das in einem übergeordneten System als d_j benutzt wird.

Trotz ihrer recht hohen Erfolgsquote besteht der Nachteil der Boosting-Methode in der Erfordernis einer sehr großen Trainingsstichprobe. Die Stichprobe sollte in drei Teile zerlegt werden, und weiterhin werden der zweite und der dritte Klassifikator nur an einer Teilmenge trainiert, an welcher der vorherige Klassifikator irrt. Ohne einen recht großen Trainingsdatensatz werden also die Trainingsmengen für d_2 und d_3 keine angemessene Größe besitzen. Drucker et al. (1994) nutzen einen Datensatz von 118.000 Instanzen beim Boosting von mehrlagigen Perzeptronen für die optische Erkennung handschriftlicher Ziffern.

ADABOOST

Freund und Schapire (1996) entwarfen eine Variante namens *AdaBoost*, kurz für *adaptive boosting*, welche dieselbe Trainingsmenge immer und immer wieder nutzt und somit auch mit einer kleineren Datenmenge auskommt. Die Klassifikatoren sollten dann aber einfach sein, damit es nicht zur Überanpassung kommt. Der AdaBoost-Algorithmus kann ebenfalls eine willkürliche Zahl an Basislernern – nicht nur drei – kombinieren.

Viele Varianten von AdaBoost wurden entworfen; hier diskutieren wir den ursprünglichen Algorithmus AdaBoost.M1 (siehe Listing 17.1). Die Idee besteht darin, die Wahrscheinlichkeiten für das Ziehen der Instanzen als eine Funktion des Fehlers zu modifizieren. Sei p_j^t die Wahrscheinlichkeit, dass das Instanzenpaar (x^t, r^t) gezogen wird, um den j-ten Basislerner zu trainieren. Anfänglich sind alle $p_1^t = 1/N$. Dann addieren wir neue Basislerner wie beschrieben, beginnend mit $j = 1$: ϵ_j bezeichnet die Fehlerrate von d_j. AdaBoost erfordert, dass $\epsilon_j < 1/2, \forall j$; ist dies nicht der

Training:
  For alle $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialisiere $p_1^t = 1/N$
  For alle Basislerner $j = 1, \ldots, L$
    Ziehe zufällig $\mathcal{X}_j$ aus $\mathcal{X}$ mit Wahrscheinlichkeiten p_j^t
    Trainiere d_j mittels $\mathcal{X}_j$
    For jedes (x^t, r^t), berechne $y_j^t \leftarrow d_j(x^t)$
    Berechne Fehlerrate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
    If $\epsilon_j > 1/2$, Then $L \leftarrow j - 1$; Beende
    $\beta_j \leftarrow \epsilon_j/(1 - \epsilon_j)$
    For jedes (x^t, r^t), verringere Wahrscheinlichkeit falls korrekt:
      If $y_j^t = r^t$ Then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
    Normalisiere Wahrscheinlichkeiten:
      $Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t/Z_j$
Testen:
  Gegeben x, berechne $d_j(x), j = 1, \ldots, L$
  Berechne Klassenausgaben, $i = 1, \ldots, K$:
    $y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$

***Listing 17.1*:** AdaBoost-Algorithmus.

Fall, dann beenden wir das Hinzufügen neuer Basislerner. Man beachte, dass diese Fehlerrate sich nicht auf das ursprüngliche Problem bezieht, sondern auf den bei Schritt j verwendeten Datensatz. Wir definieren $\beta_j = \epsilon_j/(1 - \epsilon_j) < 1$ und wir setzen $p_{j+1}^t = \beta_j p_j^t$, falls d_j eine korrekte Klassifikation für x^t erzielt; andernfalls setzen wir $p_{j+1}^t = p_j^t$. Da p_{j+1}^t Wahrscheinlichkeiten sein sollten, gibt es eine Normalisierung, bei der wir p_{j+1}^t durch $\sum_t p_{j+1}^t$ teilen, so dass sie insgesamt 1 ergeben. Das hat den Effekt, dass die Wahrscheinlichkeit einer korrekt klassifizierten Instanz verringert wird und sich die Wahrscheinlichkeit einer fehlklassifizierten Instanz erhöht. Dann wird eine neue Stichprobe derselben Größe aus der Originalstichprobe entsprechend dieser modifizierten Wahrscheinlichkeiten p_{j+1}^t mit Zurücklegen gezogen und verwendet, um d_{j+1} zu trainieren.

Dadurch wird bewirkt, dass d_{j+1} sich mehr auf durch d_j fehlklassifizierte Instanzen konzentriert. Deshalb werden Basislerner einfach gehalten und müssen nicht immer akkurat sein, da andernfalls die darauffolgende Trainingsstichprobe nur wenige Ausreißer und verrauschte Instanzen enthalten würde, die wieder und wieder wiederholt würden. Bei einem Entscheidungsbaum beispielsweise nutzt man *Entscheidungsstümpfe*, womit man Bäume bezeichnet, die nur bis zur ersten oder zweiten Ebene aufgebaut wurden. Somit ist klar, dass sie Verzerrung aufweisen werden, doch ist die Abnahme in der Varianz größer und der Gesamtfehler nimmt ab. Ein Algorithmus wie die lineare Diskriminante hat niedrige Varianz, und die Anwendung des AdaBoost-Algorithmus auf lineare Diskriminanten bringt uns gar nichts.

Ist das Training einmal vollendet, ist AdaBoost eine Methode des Voting. Für eine gegebene Instanz entscheiden alle d_j und eine gewichtete Abstimmung wird durchgeführt, bei der die Gewichte proportional zu den Genauigkeiten der Basislerner (am Trainingsdatensatz) sind: $w_j = \log(1/\beta_j)$. Freund und Schapire (1996) haben bei zweiundzwanzig Benchmark-Problemen eine verbesserte Genauigkeit erzielt, für eines eine gleichwertige Genauigkeit und bei vier Problemen verschlechterte Genauigkeit.

MARGIN

Schapire et al. (1998) erläutern, dass der Erfolg von AdaBoost an seiner Eigenschaft liegt, den *Margin* zu erhöhen. Wird dieser erhöht, dann werden die Trainingsinstanzen besser voneinander getrennt und ein Fehler ist weniger wahrscheinlich. Dadurch ähnelt AdaBoosts Ziel dem von Support-Vektor-Maschinen (Kapitel 13).

Obwohl bei AdaBoost verschiedene Basislerner leicht unterschiedliche Trainingsdatensätze verwenden, wird dieser Unterschied nicht dem Zufall überlassen wie beim Bagging, sondern ist eine Funktion des Fehlers des vorhergehenden Basislerners. Die tatsächliche Leistung beim Boosting für ein spezielles Problem hängt deutlich von den Daten und dem Basislerner ab. Es sollten ausreichend Trainingsdaten vorhanden sein und der Basislerner sollte zwar schwach, jedoch nicht zu schwach sein; außerdem ist das Boosting besonders empfänglich für Rauschen und Ausreißer.

AdaBoost ist auch für die Regression generalisiert worden: eine einfache Methode, die von Avnimelech und Intrator (1997) entwickelt wurde, überprüft, ob der Vorhersagefehler größer als ein gewisser Schwellwert ist; ist dem so, wird er als Fehler markiert und dann der eigentliche AdaBoost-Algorithmus angewandt. Bei einer anderen Version (Drucker 1997) werden Wahrscheinlichkeiten basierend auf der Größenordnung des Fehlers modifiziert, so dass Instanzen, bei denen der vorhergehende Basislerner einen schwerwiegenden Fehler begeht, eine höhere Wahrscheinlichkeit besitzen, für das Training des nächsten Basislerners gezogen zu werden. Der gewichtete Durchschnitt, oder der Median, wird verwendet, um die Vorhersagen der Basislerner zu kombinieren.

17.8 Neubetrachtung der gemischten Expertensysteme

GEMISCHTE EXPERTENSYSTEME

Beim Voting sind die Gewichte w_j über den Eingaberaum hinweg konstant. Bei den in Abschnitt 12.8 diskutierten *gemischten Expertensystemen*, die eine Erweiterung radialer Basisfunktionen und eine lokale Methode sind, gibt es ein Gatternetz, dessen Ausgaben als Gewicht beim Voting genutzt werden. Diese Architektur kann somit als eine Variante des Voting betrachtet werden, bei der die Stimmabgaben von der Eingabe abhängen und sich für verschiedene Eingaben auch unterscheiden können. Der Algorithmus des kompetitiven Lernens, der durch die gemischten

Expertenmodelle eingesetzt wird, lokalisiert die Basislerner so, dass jeder von ihnen ein Expertensystem in einem anderen Teil des Eingaberaums wird und dass sein jeweiliges Gewicht $w_j(x)$ in seiner Expertisenregion nahe 1 ist. Die finale Ausgabe ist ein gewichteter Durchschnittswert wie beim Voting:

$$y = \sum_{j=1}^{L} w_j(x) d_j \,, \tag{17.7}$$

außer dass in dem Fall sowohl die Basislerner als auch die Gewichte eine Funktion der Eingabe darstellen (siehe Abbildung 17.2).

Jacobs (1997) hat bewiesen, dass in der Architektur der gemischten Expertensysteme die Expertensysteme verzerrt aber negativ korreliert sind. Wenn das Training voranschreitet, dann verringert sich die Verzerrung und die Varianzen der Expertensysteme erhöhen sich; zur gleichen Zeit jedoch, wenn die Expertensysteme sich in verschiedenen Teilen des Eingaberaums lokalisieren, bewegen sich ihre Kovarianzen immer mehr ins Negative, wodurch sich gemäß Gleichung 17.5 die Gesamtvarianz und somit der Fehler verringert. In Abschnitt 12.8 betrachteten wir den Fall, in dem beide lineare Funktionen sind, aber auch eine nichtlineare Methode kann sowohl für die Expertensysteme als auch für das Gatter genutzt werden. Dadurch würden sich die Verzerrungen der Expertensysteme verringern, jedoch mit dem Risiko einer Erhöhung der Varianzen und einer Überanpassung.

DYNAMISCHE KLASSIFIKATORSELEKTION

Bei der *dynamischen Klassifikatorselektion* gibt es, ähnlich wie beim Gatternetz für gemischte Expertensysteme, zunächst ein System, das eine Testeingabe nimmt und die Kompetenz der Basislerner in der Umgebung der Eingabe testet. Dann wird der kompetenteste ausgewählt, um eine Ausgabe zu generieren, und diese Ausgabe wird als Gesamtausgabe betrachtet. Woods, Kegelmeyer und Bowyer (1997) finden die k-nächsten Trainingspunkte der Testeingabe, betrachten die Genauigkeiten der Basislerner auf diesen und wählen denjenigen aus, der darauf

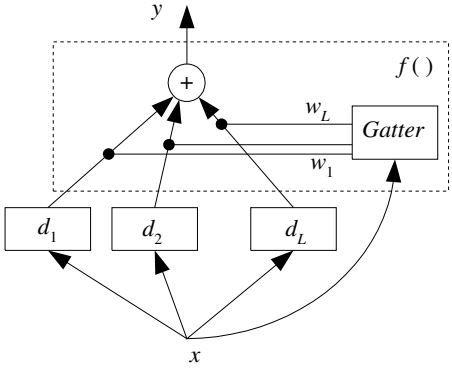


***Abb. 17.2*:** Die gemischten Expertensysteme stellen eine Variante des Voting dar, bei der Stimmabgaben, wie sie durch das Gattersystem vorgegeben werden, eine Funktion der Eingabe darstellen. Das Kombiniersystem f beinhaltet auch dieses Gattersystem.

die beste Leistung zeigt. Nur der ausgewählte Basisklassifikator muss für diese Testeingabe ausgewertet werden. Um die Varianz zu verringern, kann man auf Kosten eines höheren Rechenaufwandes ein Voting über einige wenige kompetente Basisklassifikatoren nehmen, anstatt nur einen einzigen zu verwenden.

Man beachte, dass man in einem solchen Schema sicherstellen muss, dass es für jede Region des Eingaberaums einen kompetenten Basisklassifikator gibt; das bedeutet, dass es zwischen den Basisklassifikatoren eine gewisse Aufteilung des Lernens geben sollte. Das ist der große Vorteil von gemischten Expertensystemen, nämlich dass das Gattermodell, das die Auswahl übernimmt, und die Experten-Basislerner, die ausgewählt werden, in einer gekoppelten Weise trainiert werden. Es ist nicht schwierig, eine Regressionsversion dieser dynamischen Lernerselektion abzuleiten (Übung 5).

17.9 Geschachtelte Generalisierung

GESCHACHTELTE GENERALISIERUNG

Die *geschachtelte Generalisierung* ist eine von Wolpert (1992) entwickelte Technik, welche das Voting dahingehend erweitert, dass die Art und Weise, wie die Ausgabe der Basislerner kombiniert wird, nicht linear sein muss, sondern durch ein Kombiniersystem $f(\cdot|\Phi)$ erlernt wird, welches wiederum einen Lerner darstellt, dessen Parameter Φ auch trainiert werden (siehe Abbildung 17.3):

$$y = f(d_1, d_2, \ldots, d_L|\Phi) \,. \tag{17.8}$$

Der Kombinierer lernt die korrekte Ausgabe für eine gewisse durch die Basislerner gegebene Ausgabekombination. Wir können die Kombinierfunktion nicht anhand der Trainingsdaten trainieren, da die Basislerner sich unter Umständen den Trainingsdatensatz einprägen; das Kombiniersystem sollte stattdessen eigentlich lernen, wie die Basislerner Fehler

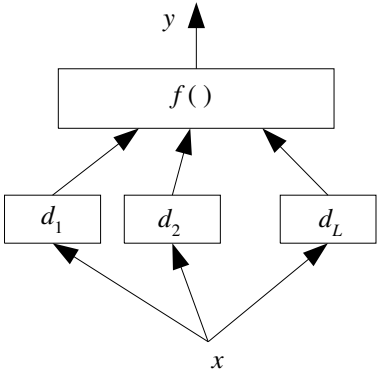


***Abb. 17.3*:** Bei der geschachtelten Generalisierung ist der Kombinierer ein weiterer Lerner und wird im Gegensatz zum Voting nicht dahingehend eingeschränkt, eine lineare Kombination sein zu müssen.

begehen. Das Schachteln ist ein Mittel für die Schätzung und Korrektur der Verzerrungen der Basislerner. Somit sollte der Kombinierer an Daten trainiert werden, die beim Training der Basislerner ungenutzt blieben.

Ist $f(\cdot|w_1, \ldots, w_L)$ ein lineares Modell mit den Nebenbedingungen $w_i \geq 0, \sum_j w_j = 1$, dann können die optimalen Gewichte durch bedingte Regression gefunden werden, aber natürlich müssen wir das nicht erzwingen; bei der Schachtelung gibt es keine Einschränkungen für die Kombinierfunktion und anders als beim Voting kann $f(\cdot)$ nichtlinear sein. Beispielsweise könnte $f(\cdot)$ ein mehrlagiges Perzeptron mit Φ als seinen Verbindungsgewichtungen sein.

Die Ausgaben der Basislerner d_j definieren einen neuen L-dimensionalen Raum, in dem die Diskriminanz-/Regressionsfunktion der Ausgabe durch die Kombinierfunktion erlernt wird.

Bei der geschachtelten Generalisierung wollen wir so unterschiedliche Basislerner wie möglich haben, so dass sie einander ergänzen, und hierfür ist es am besten, wenn sie auf unterschiedlichen Lernalgorithmen basieren. Wenn wir Klassifikatoren kombinieren, die kontinuierliche Ausgaben generieren können, beispielsweise a-posteriori-Wahrscheinlichkeiten, ist es besser, die Kombinationen anstatt harte Entscheidungen zu haben.

Wenn wir einen trainierten Kombinierer, wie wir ihn bei der Schachtelung haben, mit einer festen Regel wie beim Voting vergleichen, dann stellen wir fest, dass beide ihre Vorzüge haben: Eine trainierte Regel ist flexibler und kann eine kleinere Verzerrung haben, doch sie führt zusätzliche Parameter ein, riskiert eine höhere Varianz und benötigt zusätzliche Zeit und Daten für das Training. Außerdem ist es nicht nötig, die Ausgaben der Klassifikatoren vor der Schachtelung zu normieren.

17.10 Feinabstimmung eines Ensembles

Die Modellkombination ist keine magische Formel, die in jedem Fall eine Verringerung des Fehlers garantiert. Basislerner sollten divers und genau sein, d. h., sie sollten uns nützliche Informationen liefern. Wenn ein Basislerner keine zusätzliche Genauigkeit bringt, kann er verworfen werden. Ebenso wird einer von zwei stark korrelierten Basislernern nicht benötigt. Es ist zu bedenken, dass ein ungenauer Lerner die Gesamtgenauigkeit verschlechtern kann. So wird beispielsweise bei der Mehrheitswahl vorausgesetzt, dass für eine Eingabe mehr als die Hälfte der Klassifikatoren genau sind. Daher kann es für eine gegebene Menge von Kandidaten für Basislerner keine gute Idee sein, sie so zu verwenden, wie sie sind; stattdessen sollten wir eine Vorverarbeitung durchführen.

Wir können uns die Ausgaben unserer Basislerner als einen Merkmalsvektor für die in einem späteren Schritt erfolgende Kombination vorstellen, und wir erinnern uns, dass wir in Kapitel 6 das gleiche Problem mit

Merkmalen hatten. Einige davon können einfach nutzlos sein und andere stark korreliert. Wir können daher die gleichen Ideen, die wir im Zusammenhang mit der Merkmalsselektion und -extraktion erörtert haben, auch hier anwenden. Unser erster Ansatz besteht darin, eine Teilmenge aus der Menge der Basislerner auszuwählen, einige davon zu behalten und die übrigen zu verwerfen, während wir beim zweiten Ansatz ausgehend von den ursprünglichen Basislernern einige wenige neue, unkorrelierte Metalerner definieren.

17.10.1 Wahl einer Teilmenge des Ensembles

Ensembleselektion

Das Auswählen einer Teilmenge aus einem Ensemble von Basislernern ähnelt der Merkmalsselektion, und die möglichen Herangehensweisen sind für die *Ensembleselektion* dieselben. Wir können einen Vorwärtsansatz (auch inkrementeller Ansatz oder Wachstumsansatz) verfolgen, bei dem wir in jeder Iteration aus einer Kandidatenmenge von Basislernern denjenigen hinzufügen, der die Genauigkeit am stärksten verbessert. Alternativ können wir einen Rückwärtsansatz (auch dekrementeller Ansatz oder Pruning) verfolgen, bei dem wir in jeder Iteration denjenigen Basislerner entfernen, dessen Verschwinden zur stärksten Verbesserung führt. Eine dritte Möglichkeit besteht in einem Floatingansatz, bei dem sowohl das Hinzufügen als auch das Entfernen erlaubt ist.

Das Kombinationsschema kann eine feste Regel sein, wie etwa beim Voting, oder es kann sich um eine trainierte Schachtelung handeln. Ein solches Selektionsschema würde keine ungenauen Lerner umfassen, keine, die nicht divers genug oder korreliert sind (Caruana et al. 2004; Ruta und Gabrys 2005). Das Verwerfen der nutzlosen Lerner verringert auch die Gesamtkomplexität. Unterschiedliche Lerner können unterschiedliche Repräsentationen verwenden, und ein solcher Ansatz erlaubt es außerdem, die besten komplementären Repräsentationen zu benutzen (Demir und Alpaydın 2005). Falls wir einen Entscheidungsbaum als Kombinierer verwenden, ist zu beachten, dass er gleichzeitig als Selektor und als Kombinierer wirkt (Ulaş et al. 2009).

17.10.2 Konstruktion von Metalernern

Egal, wie wir die Lernalgorithmen, Hyperparameter, Stichprobenerhebungen oder Eingabemerkmale variieren, wir bekommen immer korrelierte Klassifikatoren (Ulaş, Yıldız und Alpaydın 2012), und es ist daher eine Nachbearbeitung nötig, um die womöglich schädliche Korrelation zu beseitigen. Eine Möglichkeit hierfür ist das bereits diskutierte Verwerfen einiger der korrelierten Klassifikatoren; eine andere besteht in der Anwendung eines Verfahrens zur Merkmalsextraktion, wobei wir von dem Raum der Ausgaben der Basislerner zu einem neuen Raum mit kleinerer Dimension übergehen, in dem wir unkorrelierte Metalerner definieren, von denen es außerdem weniger gibt.

Merz (1999) schlägt den SCANN-Algorithmus vor, der die Korrespondenzanalyse – eine Variante der Hauptkomponentenanalyse (Abschnitt 6.3) – auf den Ausgaben der Basisklassifikatoren anwendet und sie unter Verwendung des Nächster-Mittelwert-Klassifikators kombiniert. Tatsächlich können alle linearen oder nichtlinearen Verfahren der Merkmalsextraktion verwendet werden, die wir in Kapitel 6 behandelt haben, und die (vorzugsweise kontinuierliche) Ausgabe kann in jeden Lerner eingespeist werden, wie wir es bei der Schachtelung tun.

Nehmen wir an, wir haben L Lerner, von denen jeder K Ausgaben hat. Dann können wir, beispielsweise unter Verwendung der Hauptkomponentenanalyse, aus dem $K \cdot L$-dimensionalen Raum in einen neuen Raum von niedrigdimensionalen, unkorrelierten „Eigenlernern“ abbilden (Ulaş, Yıldız und Alpaydın 2012). Anschließend trainieren wir den Kombinierer in diesem neuen Raum (wobei wir eine separate Datenmenge nehmen, die für das Training der Basislerner und der Dimensionalitätsreduktion nicht verwendet wurde). Durch die Betrachtung der Koeffizienten der Eigenvektoren können wir sogar verstehen, welche Beiträge die einzelnen Basislerner leisten und so ihren Nutzen einschätzen.

Jacobs (1995) hat gezeigt, dass L abhängige Lerner genauso viel wert sind wie L' unabhängige Lerner, wobei $L' \leq L$, und das ist auch die Idee, die hier zugrunde liegt. Außerdem ist anzumerken, dass dieser Ansatz auf scharfe Entscheidungen – Verwerfen oder Behalten einer Teilmenge des Ensembles – verzichtet; stattdessen werden alle Basislerner verwendet und somit alle Informationen ausgenutzt, allerdings um den Preis, dass ein höherer Rechenaufwand nötig wird.

17.11 Kaskadierung

Die Idee bei kaskadierenden Klassifikatoren ist die, eine *Sequenz von Basisklassifikatoren* d_j zu nutzen, die hinsichtlich ihrer Speicher- bzw. Rechenzeitkomplexität oder der Kosten, der durch sie verwendeten Repräsentation, sortiert sind, so dass d_{j+1} kostspieliger ist als d_j (Kaynak und Alpaydın 2000). Die Methode des *Kaskadierens* ist eine mehrstufige Methode, und wir nutzen d_j nur, wenn alle vorhergehenden Lerner $d_k, k < j$ nicht konfident sind (siehe Abbildung 17.4). Dazu wird mit jedem Lerner eine *Konfidenz* w_j assoziiert, so dass wir sagen, dass d_j sich seiner Ausgabe sicher ist und verwendet werden kann, falls $w_j > \theta_j$, wobei $1/K < \theta_j \leq \theta_{j+1} < 1$ der Konfidenzschwellwert ist. Bei der Klassifikation ist die Konfidenzfunktion auf den höchsten a-posteriori-Wert gesetzt: $w_j \equiv \max_i d_{ji}$; dies entspricht der Strategie, die für Ablehnungen genutzt wird (Abschnitt 3.3).

KASKADIEREN

Wir nutzen den Lerner d_j, falls alle vorhergehenden Lerner nicht konfident sind:

$$y_i = d_{ji}, \text{ falls } w_j > \theta_j \text{ und } \forall k < j, w_k < \theta_k \,. \tag{17.9}$$

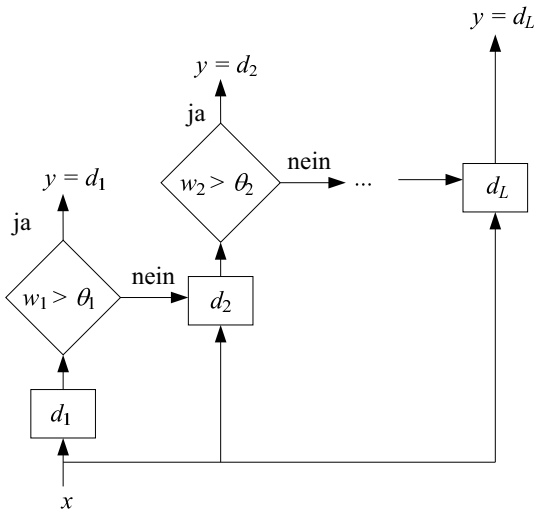


***Abb. 17.4*:** Das Kaskadieren entspricht einer mehrstufigen Methode, bei der es eine Sequenz von Klassifikatoren gibt und der jeweils nächste nur zum Einsatz kommt, wenn die vorhergehenden nicht konfident in ihrer Entscheidung sind.

Für einen gegebenen Trainingsdatensatz und beginnend mit $j = 1$ trainieren wir d_j. Dann finden wir alle Instanzen aus einem separaten Validierungsdatensatz, bei denen d_j nicht konfident ist, und diese wiederum bilden die Trainingsmenge von d_{j+1}. Man beachte, dass wir hier im Gegensatz zu AdaBoost nicht nur die fehlklassifizierten Instanzen suchen, sondern auch diejenigen, für welche der vorhergehende Basislerner nicht konfident ist. Dies deckt sowohl die Fehlklassifikationen ab als auch die Instanzen, bei denen der a-posteriori-Wert nicht hoch genug liegt; dies sind Instanzen auf der richtigen Seite der Grenze, jedoch mit einer nicht ausreichend großen Entfernung zur Diskriminante, sprich mit einem zu kleinen Margin.

Die Idee ist die, dass ein früher einfacher Klassifikator die Mehrzahl der Instanzen bearbeitet und ein komplexerer Klassifikator nur für einen kleinen Prozentsatz zum Einsatz kommt, wodurch sich die Gesamtkomplexität nicht signifikant erhöht. Dies ist gegensätzlich zu den Methoden mit mehreren Experten, wie das Voting, bei denen alle Basislerner für jede beliebige Instanz ihre Ausgabe generieren. Ist der Problemraum komplex, können einige wenige Basisklassifikatoren kaskadiert werden, wodurch die Komplexität bei jeder Stufe erhöht wird. Um die Anzahl an Basisklassifikatoren nicht zu erhöhen, werden die wenigen Instanzen, die durch keinen abgedeckt werden, so wie sie sind gespeichert und durch einen nichtparametrischen Klassifikator wie den k-NN behandelt.

Die induktive Verzerrung beim Kaskadieren ergibt sich daraus, dass die Klassen durch eine geringe Anzahl an „Regeln“ mit zunehmender Kom-

plexität erklärt werden können; zusätzlich existiert eine kleine Menge an „Ausnahmen“, die nicht durch die Regeln abgedeckt sind. Die Regeln werden durch einfache Basisklassifikatoren, beispielsweise Perzeptronen, mit zunehmender Komplexität implementiert, die allgemeine, für den gesamten Eingaberaum gültige Regeln erlernen. Ausnahmen sind lokalisierte Instanzen, die am besten durch ein nichtparametrisches Modell behandelt werden.

Das Kaskadieren steht somit zwischen den zwei Extremen der parametrischen und der nichtparametrischen Klassifikation. Erstere – zum Beispiel ein lineares Modell – findet eine einzelne Regel, die alle Instanzen abdecken sollte. Ein nichtparametrischer Klassifikator – beispielsweise k-NN – speichert die gesamte Menge an Instanzen ohne irgendeine einfache Regel zu speichern, durch welche sie erklärt werden. Durch das Kaskadieren nun wird eine Regel (oder mehrere) generiert, um einen großen Teil der Instanzen so kostengünstig wie möglich zu erklären und die verbliebenen werden als Ausnahmen abgespeichert. Dies erscheint bei vielen Lernanwendungen sinnvoll. Beispielsweise findet sich im Englischen die einfache Vergangenheitsform eines Verbs meistens, indem ein „-d“ oder „-ed“ an das Verb angehängt wird; es gibt aber auch unregelmäßige Verben – zum Beispiel „go“/„went“ – die dieser Regel nicht folgen.

17.12 Anmerkungen

Die Idee bei der Kombination von Lernern besteht darin, eine komplexe Aufgabe in einfachere Aufgaben zu zerlegen, die durch separat trainierte Basislerner gehandhabt werden. Jeder Basislerner hat seine eigene Aufgabe. Würden wir einen größeren Lerner benutzen, der alle Basislerner enthält, dann würden wir eine Überanpassung riskieren. Man betrachte zum Beispiel die Durchführung einer Wahl über drei mehrlagige Perzeptronen, von denen jedes eine verborgene Schicht besitzt. Kombinieren wir sie alle zusammen mit dem linearen Modell, das ihre Ausgaben kombiniert, dann ergibt sich ein großes mehrlagiges Perzeptron mit zwei verborgenen Schichten. Trainieren wir dieses große Modell mit der kompletten Stichprobe, dann führt dies höchstwahrscheinlich zu einer Überanpassung. Trainieren wir die drei mehrlagigen Perzeptronen separat, beispielsweise durch Anwendung von ECOC, Bagging, usw., dann ist es, als ob wir eine geforderte Ausgabe für die verborgenen Knoten der zweiten Schicht des großen mehrlagigen Perzeptrons definieren. Dadurch wird dem allumfassenden Lerner eine Einschränkung hinsichtlich dessen auferlegt, was er zu lernen hat, wodurch das Lernen vereinfacht wird.

Ein Nachteil beim Kombinieren besteht darin, dass das kombinierte System nicht interpretierbar ist. Beispielsweise sind zwar Entscheidungsbäume interpretierbar; Bäume, auf die der Bagging- oder Boosting-Algorithmus angewandt wurde, sind dies aber nicht. Fehlerkorrekturcodes mit ihren Gewichten als $-1/0/+1$ erlauben ein gewisses Maß

an Interpretierbarkeit. Mayoraz und Moreira (1997) diskutieren inkrementelle Methoden zum Erlernen der Fehlerkorrekturcodes, bei denen Basislerner je nach Bedarf hinzugefügt werden. Allwein, Schapire und Singer (2000) befassen sich mit diversen Methoden für die Codierung von Mehrklassenproblemen als Zweiklassenprobleme. Alpaydın und Mayoraz (1999) betrachten die Anwendung der ECOC, bei der lineare Basislerner kombiniert werden, um nichtlineare Diskriminanten zu erhalten, und sie entwickeln außerdem Methoden, um die ECOC-Matrix anhand von Daten zu lernen.

Der älteste und intuitivste Ansatz ist das Voting. Xu, Krzyzak und Suen (1992) liefern eine frühe Studie. Benedikttson und Swain (1992) erwägen Varianten des Voting für die Kombination multipler Quellen. Kittler et al. (1998) bieten eine neuzeitliche Betrachtung des Voting und diskutieren außerdem eine Applikation, bei der multiple Repräsentationen kombiniert werden. Die Aufgabe besteht in der Personenidentifikation mit Hilfe dreier Darstellungen: ein Frontalbild des Gesichts, ein Bild des Gesichtsprofils und die Stimme. Die Fehlerrate beim Voting mit diesem Modell ist niedriger als die Fehlerraten bei Verwendung einer einzelnen Repräsentation. Eine weitere Anwendung findet sich bei Alimoğlu und Alpaydın (1997), wo zum Zwecke der verbesserten Erkennung handschriftlicher Ziffern zwei Informationsquellen kombiniert werden: eine bildet die Daten zur temporalen Stiftbewegung beim Schreiben des Zeichens auf einem bewegungsempfindlichen Pad, und die andere Quelle findet sich durch das statische, zweidimensionale Bitmap-Bild, nachdem das Zeichen geschrieben wurde. In jener Applikation haben die beiden Klassifikatoren bei Nutzung einer der beiden Repräsentationen einen Fehler von zirka 5 Prozent; durch die Kombination der beiden reduziert sich die Rate auf 3 Prozent. Es zeigt sich außerdem, dass die kritische Phase im Entwurf der komplementären Lerner und/oder Repräsentationen besteht; die Art und Weise, wie sie kombiniert werden, erweist sich nicht als dermaßen kritisch.

Biometrie

Das Kombinieren verschiedener Modalitäten wird in der *Biometrie* angewendet, wobei das Ziel die Authentifikation mittels unterschiedlicher Eingabequellen – wie Fingerabdruck, Unterschrift und Gesichtsmerkmale – ist. In einem solchen Fall nutzen verschiedene Klassifikatoren diese Modalitäten separat, und anschließend werden ihre Entscheidungen kombiniert. Dieses Vorgehen verbessert zum einen die Genauigkeit und erschwert zum anderen Manipulationen.

Noble (2004) unterscheidet drei Typen von Kombinationsstrategien, wenn wir Informationen haben, die aus unterschiedlichen Quellen stammen und in unterschiedlichen Repräsentationen vorliegen:

- *Frühe Integration* bedeutet, alle diese Eingaben zu verknüpfen und daraus einen einzigen Vektor zu bilden, der dann in einen einzigen Klassifikator eingespeist wird. Wir haben weiter vorn erörtert, warum das keine besonders gute Idee ist.

- Bei der *späten Integration,* mit der wir uns in diesem Kapitel beschäftigt haben, werden unterschiedliche Eingaben in separaten Klassifikatoren verarbeitet. Deren Ausgaben werden dann durch Voting, Schachtelung oder eine der anderen hier vorgestellten Methoden kombiniert.
- Kernel-Algorithmen, die wir in Kapitel 13 diskutiert haben, gestatten einen weiteren Ansatz der Integration, den Noble (2004) als *intermediäre Integration* bezeichnet, da er einen Mittelweg zwischen früher und später Integration darstellt. Dies ist das *Multiple-Kernel-Lernen* (siehe Abschnitt 13.8), bei dem es einen einzelnen Klassifikator mit Kernel-Maschine gibt, der aber für verschiedene Eingaben unterschiedliche Kernel verwendet. Die Kombination erfolgt in diesem Fall weder im Eingaberaum wie bei der frühen Integration, noch im Entscheidungsraum wie bei der späten Integration, sondern im Raum der Basisfunktionen, die die Kernel definieren. Für die verschiedenen Quellen gibt es verschiedene Ähnlichkeitsmaße, die jeweils von den Kerneln berechnet werden. Diese werden von dem Klassifikator akkumuliert und verwendet.

Multiple-Kernel-Lernen

Einige Ensemblemethoden wie etwa das Voting ähneln der Bayesschen Mittelung (Kapitel 16). Wenn wir zum Beispiel Bagging anwenden und das gleiche Modell auf verschiedenen neu zusammengestellten Trainingsmengen trainieren, dann können wir Letztere als Stichproben aus einer a-posteriori-Verteilung auffassen. Andere Kombinationsmethoden wie gemischte Experten oder die Schachtelung gehen weit über das Mitteln über Parameter oder Modelle hinaus.

Wenn wir unterschiedliche Ansichten oder Repräsentationen kombinieren, dann ist die simple Verknüpfung nicht wirklich eine gute Idee. Eine interessante Möglichkeit wäre dagegen, so etwas wie eine kombinierte Dimensionalitätsreduktion durchzuführen. Wir können ein generatives Modell betrachten (Abschnitt 14.3), bei dem wir annehmen, dass es eine Menge von latenten Faktoren gibt, welche diese multiplen Ansicht parallel generieren. Von den beobachteten Ansichten können wir dann zurück in den latenten Raum gehen und dort die Klassifikation ausführen (Chen et al. 2012).

Die Kombination multipler Lerner ist schon seit den frühen 1990er-Jahren ein beliebtes Thema auf dem Gebiet des maschinellen Lernens und die Forschung wurde seitdem immer weiter vorangetrieben. Kuncheva (2004) diskutiert verschiedene Aspekte der Kombination von Klassifikatoren, und das Buch enthält auch einen Abschnitt über die Kombination von mehreren Clusterergebnissen.

Entscheidungsbäume mit AdaBoost werden zu den besten maschinellen Lernalgorithmen gezählt. Es existieren auch Versionen von AdaBoost, bei denen der nachfolgende Basislerner an Restelementen des vorhergehenden Basislernerns trainiert wird (Hastie, Tibshirani und Friedman 2001). In

letzter Zeit wurde angemerkt, dass Ensembles die Genauigkeit nicht immer verbessern, und die Forschung richtet ihr Augenmerk zunehmend auf die Kriterien, die von einem guten Ensemble erfüllt werden sollten bzw. wie ein solches zu bilden ist. Einen Überblick über die Bedeutung der Diversität in Ensembles bietet Kuncheva 2005.

17.13 Übungen

1. Wenn jeder Basislerner unabhängig und identisch verteilt ist und mit der Wahrscheinlichkeit $p > 1/2$ korrekt liegt, wie lautet dann die Wahrscheinlichkeit dafür, dass eine Mehrheitswahl über L Klassifikatoren die korrekte Antwort liefert?

   LÖSUNG: Sie ist gegeben durch eine Binomialverteilung (siehe Abbildung 17.5):

$$P(X \geq \lfloor L/2 \rfloor + 1) = \sum_{i=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{i} p^i (1-p)^{L-i} .$$

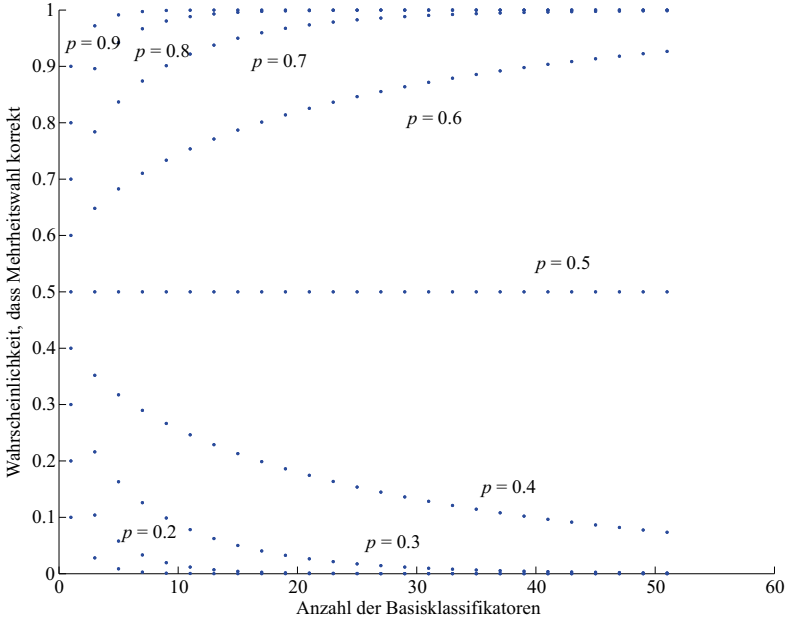


Abb. 17.5: Wahrscheinlichkeit, dass eine Mehrheitswahl korrekt ist, als Funktion der Anzahl der Basislerner für verschiedene p. Die Wahrscheinlichkeit wächst nur für $p > 0{,}5$.

2. Was wäre der Effekt der Verwendung von L-facher Kreuzvalidierung statt der Bootstrap-Methode, um beim Bagging L Trainingsdatensätze zu generieren?

3. Entwerfen Sie einen inkrementellen Algorithmus zum Erlernen von Fehlerkorrekturcodes, bei dem neue Zweiklassenprobleme je nach Bedarf hinzugefügt werden, um das Mehrklassenproblem besser lösen zu können.

4. In einer gemischten Expertenarchitektur können wir verschiedene Experten haben, die unterschiedliche Repräsentationen für die Eingaben verwenden. Wie können wir in einem solchen Fall das Gatternetz entwerfen?

5. Schlagen Sie einen Auswahlalgorithmus für einen dynamischen Regressor vor.

6. Worin besteht der Unterschied zwischen dem Voting und der Schachtelung bei Verwendung eines linearen Perzeptrons als Kombinierfunktion?

   LÖSUNG: Wenn das Voting-System ebenfalls trainiert ist, besteht der einzige Unterschied darin, dass die Gewichte bei der Schachtelung nicht positiv sein oder sich zu 1 addieren müssen, und es gibt außerdem einen Term für die Verzerrung. Natürlich zeigt sich der Hauptvorteil der Schachtelung dann, wenn der Kombinierer nichtlinear ist.

7. Warum fordern wir beim Kaskadieren, dass $\theta_{j+1} \geq \theta_j$?

   LÖSUNG: Instanzen, auf denen die Konfidenz kleiner als θ_j ist, wurden durch d_j bereits herausgefiltert; wir fordern, dass der Schwellwert größer wird, so dass wir größere Konfidenzen erreichen können.

8. Um das Kaskadieren bei der Regression verwenden zu können, sollte ein Regressor während des Testens in der Lage sein zu sagen, ob er konfident in Bezug auf seine Ausgabe ist. Wie können wir das implementieren?

9. Wie können wir die Ergebnisse mehrerer Clusterlösungen kombinieren?

   LÖSUNG: Die einfachste Möglichkeit ist die folgende. Wir betrachten dazu zwei Trainingsinstanzen. Jede Clusterlösung platziert die beiden entweder im gleichen Cluster oder nicht, was wir mit 1 bzw. 0 bezeichnen. Der Mittelwert dieser Zähler über alle Clusterlösungen entspricht der Gesamtwahrscheinlichkeit, dass die beiden zum gleichen Cluster gehören (Kuncheva 2004).

10. In Abschnitt 17.10 haben wir erörtert, dass ein Entscheidungsbaum, den wir bei der Schachtelung als Kombinierer verwenden, sowohl als Selektor als auch als Kombinierer wirkt. Was sind die anderen Vor- und Nachteile?

    Lösung: Ein Baum benutzt nur eine Teilmenge der Klassifikatoren und nicht alle. Die Verwendung eines Baums ist schnell, und wir müssen nur die Knoten auf unserem Pfad auswerten, der kurz sein kann. Siehe Ulaş et al 2009 für eine ausführlichere Darstellung. Der Nachteil ist, dass der Kombinierer keine Kombinationen von Klassifikatoren betrachten kann (vorausgesetzt, der Baum ist univariat). Die Verwendung einer Teilmenge kann ebenfalls von Nachteil sein; wir erhalten nicht die Redundanz, die wir brauchen, falls einige Klassifikatoren fehlerhaft sind.

17.14 Literaturangaben

Alimoğlu, F., and E. Alpaydın. 1997. „Combining Multiple Representations and Classifiers for Pen-Based Handwritten Digit Recognition." In *Fourth International Conference on Document Analysis and Recognition,* 637–640. Los Alamitos, CA: IEEE Computer Society.

Allwein, E. L., R. E. Schapire, and Y. Singer. 2000. „Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers." *Journal of Machine Learning Research* 1: 113–141.

Alpaydın, E. 1997. „Voting over Multiple Condensed Nearest Neighbors." *Artificial Intelligence Review* 11: 115–132.

Alpaydın, E., and E. Mayoraz. 1999. „Learning Error-Correcting Output Codes from Data." In *Ninth International Conference on Artificial Neural Networks,* 743–748. London: IEE Press.

Avnimelech, R., and N. Intrator. 1997. „Boosting Regression Estimators." *Neural Computation* 11: 499–520.

Breiman, L. 1996. „Bagging Predictors." *Machine Learning* 26: 123–140.

Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes. 2004. „Ensemble Selection from Libraries of Models." In *Twenty-First International Conference on Machine Learning,* ed. C. E. Brodley, 137–144. New York: ACM.

Chen, N., J. Zhu, F. Sun, and E. P. Xing. 2012. „Large-Margin Predictive Latent Subspace Learning for Multiview Data Analysis." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34: 2365–2378.

Demir, C., and E. Alpaydın. 2005. „Cost-Conscious Classifier Ensembles." *Pattern Recognition Letters* 26: 2206–2214.

Dietterich, T. G., and G. Bakiri. 1995. „Solving Multiclass Learning Problems via Error-Correcting Output Codes.“ *Journal of Artificial Intelligence Research* 2: 263–286.

Drucker, H. 1997. „Improving Regressors using Boosting Techniques.“ In *Fourteenth International Conference on Machine Learning,* ed. D. H. Fisher, 107–115. San Mateo, CA: Morgan Kaufmann.

Drucker, H., C. Cortes, L. D. Jackel, Y. Le Cun, and V. Vapnik. 1994. „Boosting and Other Ensemble Methods.“ *Neural Computation* 6: 1289–1301.

Freund, Y., and R. E. Schapire. 1996. „Experiments with a New Boosting Algorithm.“ In *Thirteenth International Conference on Machine Learning,* ed. L. Saitta, 148–156. San Mateo, CA: Morgan Kaufmann.

Hansen, L. K., and P. Salamon. 1990. „Neural Network Ensembles.“ *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12: 993–1001.

Hastie, T., R. Tibshirani, and J. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer.

Ho, T. K. 1998. „The Random Subspace Method for Constructing Decision Forests.“ *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20: 832–844.

Jacobs, R. A. 1995. „Methods for Combining Experts' Probability Assessments.“ *Neural Computation* 7: 867–888.

Jacobs, R. A. 1997. „Bias/Variance Analyses for Mixtures-of-Experts Architectures.“ *Neural Computation* 9: 369–383.

Jain, A., K. Nandakumar, and A. Ross. 2005. „Score Normalization in Multimodal Biometric Systems.“ *Pattern Recognition* 38: 2270–2285.

Kaynak, C., and E. Alpaydın. 2000. „MultiStage Cascading of Multiple Classifiers: One Man's Noise is Another Man's Data.“ In *Seventeenth International Conference on Machine Learning,* ed. P. Langley, 455–462. San Francisco: Morgan Kaufmann.

Kittler, J., M. Hatef, R. P. W. Duin, and J. Matas. 1998. „On Combining Classifiers.“ *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20: 226–239.

Kuncheva, L. I. 2004. *Combining Pattern Classifiers: Methods and Algorithms.* Hoboken, NJ: Wiley.

Kuncheva, L. I. 2005. Special issue on Diversity in Multiple Classifier Systems. *Information Fusion* 6: 1–115.

Mayoraz, E., and M. Moreira. 1997. „On the Decomposition of Polychotomies into Dichotomies." In *Fourteenth International Conference on Machine Learning,* ed. D. H. Fisher, 219–226. San Mateo, CA: Morgan Kaufmann.

Merz, C. J. 1999. „Using Correspondence Analysis to Combine Classifiers." *Machine Learning* 36: 33–58.

Noble, W. S. 2004. „Support Vector Machine Applications in Computational Biology." In *Kernel Methods in Computational Biology,* ed. B. Schölkopf, K. Tsuda, and J.-P. Vert, 71–92. Cambridge, MA: MIT Press.

Özen, A., M. Gönen, E. Alpaydın, and T. Haliloğlu. 2009. „Machine Learning Integration for Predicting the Effect of Single Amino Acid Substitutions on Protein Stability." *BMC Structural Biology* 9 (66): 1–17.

Perrone, M. P. 1993. „Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure." Ph.D. thesis, Brown University.

Ruta, D., and B. Gabrys. 2005. „Classifier Selection for Majority Voting." *Information Fusion* 6: 63–81.

Schapire, R. E. 1990. „The Strength of Weak Learnability." *Machine Learning* 5: 197–227.

Schapire, R. E., Y. Freund, P. Bartlett, and W. S. Lee. 1998. „Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods." *Annals of Statistics* 26: 1651–1686.

Ulaş, A., M. Semerci, O. T. Yıldız, and E. Alpaydın. 2009. „Incremental Construction of Classifier and Discriminant Ensembles." *Information Sciences* 179: 1298–1318.

Ulaş, A., O. T. Yıldız, and E. Alpaydın. 2012. „Eigenclassifiers for Combining Correlated Classifiers." *Information Sciences* 187: 109–120.

Wolpert, D. H. 1992. „Stacked Generalization." *Neural Networks* 5: 241–259.

Woods, K., W. P. Kegelmeyer Jr., and K. Bowyer. 1997. „Combination of Multiple Classifiers Using Local Accuracy Estimates." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19: 405–410.

18 Bestärkendes Lernen

Beim bestärkenden Lernen ist der Lerner ein entscheidungstreffender Agent, der in einer Umgebung Handlungen ausführt und Belohnung (oder Bestrafung) für seine Aktionen beim Versuch, das Problem zu lösen, erfährt. Nach einer Menge an Versuch-und-Irrtum-Durchläufen sollte er die beste Vorgehensweise lernen, welche der Sequenz an Aktionen entspricht, durch welche die Gesamtbelohnung maximiert wird.

18.1 Einführung

Nehmen wir an, wir wollen eine Maschine bauen, die das Schachspielen lernt. In dem Fall können wir aus zwei Gründen keinen überwachten Lerner wählen: zum einen ist es sehr kostspielig, einen Lehrer zu verwenden, der uns durch viele Spiele führt und uns für jede Stellung den besten Zug anzeigt. Zum anderen gibt es in vielen Fällen keinen besten Zug; die Qualität eines Zuges hängt von den Folgezügen ab. Ein einzelner Zug zählt nicht; eine Sequenz von Zügen ist dann gut, wenn wir nach dem Spielen dieser Züge das Spiel gewinnen. Die einzige verfügbare Rückmeldung erhalten wir am Spielende, wenn wir die Partie gewinnen oder verlieren.

Ein weiteres Beispiel ist ein Roboter, der in ein Labyrinth gesetzt wird. Der Roboter kann sich in eine der vier Himmelsrichtungen bewegen und sollte eine Sequenz an Bewegungen durchführen, um den Ausgang zu erreichen. Solange sich der Roboter im Labyrinth aufhält, gibt es keine Rückmeldungen und der Roboter probiert viele Bewegungen aus, bis er den Ausgang erreicht; erst dann erhält er seine Belohnung. In diesem Szenario gibt es keinen Gegner, jedoch können wir Vorlieben hinsichtlich kürzerer Abläufe haben, das heißt, wir spielen gegen die Zeit.

Diese zwei Applikationen haben einiges gemeinsam: es existiert ein Entscheidungsträger, bezeichnet als *Agent*, der in eine *Umwelt* gesetzt wird (siehe Abbildung 18.1). Beim Schach ist der Spieler der Entscheidungsträger und die Umgebung wird durch das Spielbrett repräsentiert; im zweiten Szenario ist das Labyrinth die Umgebung des Roboters. Zu jeder Zeit befindet sich die Umwelt in einem gewissen *Zustand*, der einer aus einer Menge an möglichen Zuständen ist – zum Beispiel der Spielstand auf dem Brett oder die Position des Roboters im Labyrinth. Der Entscheidungsträger verfügt über eine Menge an möglichen *Aktionen*: zulässiges Bewegen der Spielfiguren auf dem Schachbrett, das Vorwärtsbewegen des

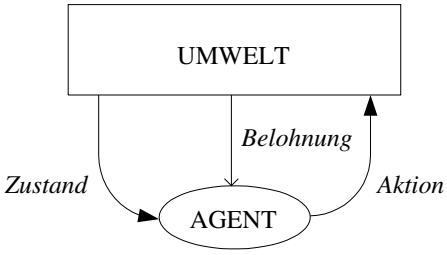


Abb. 18.1: Der Agent interagiert mit einer Umwelt. In jedem Zustand der Umwelt führt der Agent eine Aktion aus, die den Zustand verändert und eine Belohnung (oder Bestrafung) liefert.

Roboters in mögliche Richtungen, ohne gegen die Wände zu laufen, und so weiter. Sobald eine Aktion gewählt und durchgeführt wird, ändert sich der Zustand. Die Lösung der Aufgabe erfordert eine Sequenz von Aktionen; wir erhalten selten Rückmeldungen in Form einer *Belohnung*, und im Allgemeinen erst, wenn die komplette Sequenz ausgeführt wurde. Die Belohnung definiert das Problem und ist notwendig, wenn wir von einem *lernenden* Agenten sprechen wollen. Der lernende Agent lernt die beste Sequenz an Aktionen, welche die maximale kumulative Belohnung erzielen. All das ergibt die Ausgangssituation beim *bestärkenden Lernen*.

Kritiker

Das bestärkende Lernen unterscheidet sich von den Lernmethoden, die wir bereits behandelt haben, in vielerlei Hinsicht: es wird als „Lernen mit einem Kritiker“ bezeichnet, im Gegensatz zum Lernen mit einem Lehrer, wie wir es beim überwachten Lernen vorfinden. Ein *Kritiker* unterscheidet sich von einem Lehrer dahingehend, dass er uns nicht sagt, was wir zu tun haben, sondern nur, wie gut wir in der Vergangenheit abgeschnitten haben; der Kritiker informiert uns niemals im Voraus. Die Rückmeldungen vom Kritiker sind rar, und wenn einmal eine kommt, dann kommt sie spät. Dies führt zum Problem, die eventuell erhaltene Belohnung der richtigen Handlung zuzuweisen. Nachdem wir viele Aktionen vollzogen und die Belohnung erhalten haben, möchten wir unsere individuellen Aktionen der Vergangenheit bewerten und diejenigen Züge herausfinden, die uns zum Erhalt der Belohnung geführt haben, so dass wir sie festhalten und später wieder abrufen können. Wie wir in Kürze sehen werden, besteht ein Programm des bestärkenden Lernens daraus, dass es lernt, einen *internen Wert* für die zwischenzeitlichen Zustände oder Aktionen zu generieren, der ausdrückt, wie gut sie uns in Richtung Ziel weiterbringen und uns zur wahren Belohnung führen. Sobald solch ein interner Belohnungsmechanismus erlernt ist, kann der Agent einfach die lokalen Aktionen ausführen, um die Belohnung zu maximieren.

Erkennen richtiger Aktionen

Die Lösung für das gestellte Problem erfordert eine *Sequenz* von Aktionen, und aus dieser Perspektive erinnern wir uns an die Markov-Modelle, die wir in Kapitel 15 besprochen haben. Tatsächlich nutzen wir einen Markov-Entscheidungsprozess, um den Agenten zu modellieren. Der Unterschied besteht darin, dass es im Falle von Markov-Modellen einen externen Prozess gibt, der eine Sequenz an Signalen generiert, beispielsweise die Sprache, die wir beobachten und modellieren. Im hier vorliegenden Fall

jedoch generiert der Agent die Sequenz an Aktionen. Bisher haben wir außerdem zwischen beobachtbaren und Hidden-Markov-Modellen unterschieden, bei denen die Zustände beobachtet werden bzw. verborgen bleiben (und hergeleitet werden sollten). Auf ähnliche Weise liegt hier in manchen Fällen ein teilweise beobachtbarer Markov-Entscheidungsprozess vor, in denen der Agent seinen eigenen Zustand nicht genau kennt, ihn jedoch mit einiger Ungewissheit durch Beobachtungen mittels Sensoren ableiten sollte. Im Falle eines sich in einem Raum bewegenden Roboters kennt der Roboter unter Umständen weder seine genaue Position im Raum, noch die exakte Position von Hindernissen, noch das Ziel und sollte dennoch Entscheidungen mit Hilfe eines begrenzten durch die Kamera bereitgestellten Bildes treffen.

18.2 Fälle mit einem Zustand: K-armiger Bandit

K-armiger Bandit

Wir beginnen mit einem einfachen Beispiel. Der *K-armige Bandit* ist ein hypothetischer Glücksspielautomat mit K Hebeln. Die Aktion besteht darin, einen der Hebel zu wählen und zu betätigen, und wir gewinnen einen bestimmten Geldbetrag, welcher der Belohnung (der Aktion), die dem Hebel zugehört, entspricht. Die Aufgabe besteht darin zu entscheiden, welcher Hebel zu betätigen ist, um die Belohnung zu maximieren. Hierbei handelt es sich um ein Klassifikationsproblem, bei dem wir eins der K wählen. Würde es sich um überwachtes Lernen handeln, so würde uns der Lehrer die korrekte Klasse mitteilen, sprich, den Hebel, welcher zu maximalem Gewinn führt. Im Fall des bestärkenden Lernens können wir lediglich verschiedene Hebel probieren und uns den besten merken. Dies ist ein vereinfachtes Problem des bestärkenden Lernens, da es nur einen Zustand, also nur einen Glücksspielautomaten gibt und wir uns nur hinsichtlich der Aktion entscheiden müssen. Ein weiterer Grund dafür, dass es sich um ein vereinfachtes Szenario handelt, ist der, dass wir sofort eine Belohnung erhalten, nachdem eine einzelne Aktion durchgeführt wurde; die Belohnung erreicht uns ohne zeitliche Verzögerung, so dass wir sofort den Wert unserer Handlung erkennen.

Sei $Q(a)$ der Wert für Aktion a. Zu Beginn gilt: $Q(a) = 0$ für alle a. Probieren wir Aktion a, erhalten wir die Belohnung $r_a \geq 0$. Wenn Belohnungen deterministisch sind, dann erhalten wir immer dieselbe Belohnung r_a für jede beliebige Ausführung von a, und in diesem Fall können wir genauso gut $Q(a) = r_a$ setzen. Wollen wir die Belohnung weiter ausschöpfen, so können wir, sobald wir eine Aktion a gefunden haben, für die $Q(a) > 0$ gilt, diese Aktion immer wieder wählen und erhalten bei jeder Durchführung r_a. Allerdings ist es durchaus möglich, dass es einen anderen Hebel mit einer höheren Belohnung gibt, so dass wir alle Möglichkeiten erforschen sollten.

Wir können verschiedene Aktionen wählen und speichern $Q(a)$ für alle a. Immer, wenn wir eine Belohnung ausschöpfen wollen, können wir die Aktion mit dem maximalen Wert wählen, sprich

$$\text{wähle } a^*, \text{ falls } Q(a^*) = \max_a Q(a) \,. \tag{18.1}$$

Wenn Belohnungen nicht deterministisch sind, sondern stochastisch, dann erhalten wir mit jeder Durchführung derselben Aktion eine unterschiedliche Belohnung. Die Höhe der Belohnung definiert sich durch die Wahrscheinlichkeitsverteilung $p(r|a)$. In solch einem Fall definieren wir $Q_t(a)$ als die Schätzung des Wertes von Aktion a zum Zeitpunkt t. Sie ist ein Durchschnitt aus allen erhaltenen Belohnungen, wenn Aktion a vor dem Zeitpunkt t ausgeführt wurde. Eine online Aktualisierung lässt sich definieren als

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta[r_{t+1}(a) - Q_t(a)] \tag{18.2}$$

mit $r_{t+1}(a)$ als erhaltene Belohnung, nachdem Aktion a zum $(t+1)$-ten Zeitpunkt ausgeführt wurde.

Man beachte, dass Gleichung 18.2 die *Deltaregel* ist, die wir zu vielen Gelegenheiten in den vergangen Kapiteln genutzt haben: η ist der Lernfaktor (zeitlich schrittweise zu Zwecken der Konvergenz verringert), r_{t+1} ist die gewünschte Ausgabe und $Q_t(a)$ ist die gegenwärtige Vorhersage. $Q_{t+1}(a)$ ist der *Erwartungswert* von Aktion a zum Zeitpunkt $t+1$ und konvergiert zum Mittelwert von $p(r|a)$, wenn sich t erhöht.

Das komplette Problem des bestärkenden Lernens generalisiert diesen einfachen Fall auf vielfache Weise. Erstens haben wir mehrere Zustände. Dies entspricht dem Vorhandensein von mehreren Glücksspielautomaten mit verschiedenen Belohnungswahrscheinlichkeiten, $p(r|s_i, a_j)$, und wir müssen $Q(s_i, a_j)$ lernen, was dem Wert der Durchführung von Aktion a_j im Zustand s_i entspricht. Zweitens beeinflussen die Aktionen nicht nur die Belohnung, sondern auch den Folgezustand, und wir gehen von einem Zustand in den nächsten über. Drittens sind die Belohnungen verzögert und wir müssen in der Lage sein, sofortige Werte anhand verzögerter Belohnungen zu schätzen.

18.3 Elemente des bestärkenden Lernens

Der lernende Entscheidungsträger wird als *Agent* bezeichnet. Der Agent interagiert mit der *Umwelt*, welche alles außerhalb des Agenten umfasst. Der Agent besitzt Sensoren, um eine Entscheidung hinsichtlich seines *Zustands* in der Umwelt zu treffen und führt eine *Aktion* aus, die seinen Zustand modifiziert. Wenn der Agent eine Aktion durchführt, wird er durch die Umwelt *belohnt* (oder *bestraft*). Die Zeit ist diskret mit $t = 0, 1, 2, \ldots$ und $s_t \in \mathcal{S}$ bezeichnet den Zustand des Agenten zum

Zeitpunkt t, wobei $\mathcal{S}$ die Menge aller möglichen Zustände bezeichnet. $a_t \in \mathcal{A}(s_t)$ steht für die Aktion, die der Agent zum Zeitpunkt t ausführt, wobei $\mathcal{A}(s_t)$ für die Menge aller möglichen Aktionen im Zustand s_t steht. Wenn der Agent im Zustand s_t die Aktion a_t ausführt, dann läuft die Zeit weiter, die Belohnung $r_{t+1} \in \mathbb{R}$ wird empfangen und der Agent geht in den nächsten Zustand, s_{t+1}, über. Das Problem wird mit Hilfe eines *Markovschen Entscheidungsprozesses* (engl. *Markov decision process*, MDP) modelliert. Die Belohnung und der Folgezustand werden stichprobenartig aus ihren jeweiligen Verteilungen, $p(r_{t+1}|s_t, a_t)$ und $P(s_{t+1}|s_t, a_t)$, entnommen. Man beachte, dass wir hier ein *Markov-System* haben, in dem Zustand und Belohnung im nächsten Zeitschritt nur vom gegenwärtigen Zustand und der Aktion abhängen. Bei einigen Applikationen sind die Belohnung und der Folgezustand deterministisch, und für einen gewissen Zustand und eine gewisse gewählte Aktion gibt es einen möglichen Belohnungswert und Folgezustand.

MARKOVSCHER ENTSCHEIDUNGSPROZESS

In Abhängigkeit von der Anwendung kann ein bestimmter Zustand als Ausgangszustand festgelegt werden, und bei einigen Applikationen gibt es auch einen absorbierenden terminalen (Ziel-)Zustand, an dem die Suche endet; alle Aktionen in diesem terminalen Zustand gehen in sich selbst über mit der Wahrscheinlichkeit 1 und ohne jegliche Belohnung. Die Sequenz an Aktionen vom Start zum terminalen Zustand bezeichnet man als *Episode* oder *Versuch*.

EPISODE

Die *Taktik* π, definiert das Verhalten des Agenten und ist eine Abbildung von den Zuständen der Umwelt auf Aktionen: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Die Taktik definiert die in einem beliebigen Zustand s_t auszuführende Handlung: $a_t = \pi(s_t)$. Der *Wert* einer Taktik π, $V^\pi(s_t)$, ist die erwartete kumulative Belohnung, die der Agent beim Befolgen der Taktik erhält, beginnend mit Zustand s_t.

TAKTIK

Im Modell des *finiten Horizonts*, auch als *episodisches* Modell bekannt, versucht der Agent die erwartete Belohnung für die nächsten T Schritte zu maximieren:

FINITER HORIZONT

$$V^\pi(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T}] = E\left[\sum_{i=1}^{T} r_{t+i}\right] . \qquad (18.3)$$

Gewisse Aufgaben bestehen fortwährend und es besteht kein a priori festgelegtes Limit für die Episode. Im Modell des *infiniten Horizonts* gibt es kein Limit für die Sequenz, jedoch werden zukünftige Belohnungen diskontiert:

INFINITER HORIZONT

$$V^\pi(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] , \qquad (18.4)$$

wobei $0 \leq \gamma < 1$ die *Skontorate* ist, um den Gewinn endlich zu halten. Falls $\gamma = 0$, dann zählt lediglich die sofortige Belohnung. Wenn γ sich

SKONTORATE

1 annähert, dann zählen ferner in der Zukunft liegende Belohnungen mehr, und wir sprechen davon, dass der Agent weitsichtiger wird. γ ist kleiner als 1, da es im Allgemeinen ein zeitliches Limit für die Sequenz an Aktionen gibt, die zur Lösung der Aufgabe benötigt werden. Der Agent könnte ein batteriebetriebener Roboter sein. Wir ziehen es also vor, die Belohnung lieber früher als später zu erhalten, weil wir nicht sicher sind, wie lange wir im System überleben werden.

OPTIMALE TAKTIK

Für jede Taktik π existiert ein $V^\pi(s_t)$ und wir wollen die *optimale Taktik* π^* finden, so dass

$$V^*(s_t) = \max_\pi V^\pi(s_t), \forall s_t \,. \tag{18.5}$$

Bei einigen Anwendungen, beispielsweise bei Steuerungssystemen, ziehen wir es vor, statt mit den Werten von Zuständen, $V(s_t)$, lieber mit den Werten von Zustand-Aktions-Paaren, $Q(s_t, a_t)$, zu arbeiten. $V(s_t)$ gibt an, wie günstig es für den Agenten ist, sich im Zustand s_t zu befinden, wohingegen $Q(s_t, a_t)$ angibt, wie vorteilhaft es ist, Aktion a_t im Zustand s_t auszuführen. Wir definieren $Q^*(s_t, a_t)$ als den Wert der Aktion, das heißt, die erwartete kumulative Belohnung für Aktion a_t, wenn sie im Zustand s_t ausgeführt und daraufhin die optimale Taktik befolgt wird. Der Wert eines Zustands ist gleich dem Wert der besten möglichen Aktion:

$$\begin{aligned}
V^*(s_t) &= \max_{a_t} Q^*(s_t, a_t) \\
&= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1}\right] \\
&= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \\
V^*(s_t) &= \max_{a_t}\Big(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1})\Big). \qquad (18.6)
\end{aligned}$$

Wir gehen in jeden möglichen Folgezustand s_{t+1} mit der Wahrscheinlichkeit $P(s_{t+1}|s_t, a_t)$ über, und wenn wir von dort an unter Befolgung der optimalen Taktik fortfahren, dann ist die erwartete kumulative Belohnung $V^*(s_{t+1})$. Wir summieren über alle solchen möglichen Folgezustände, und wir diskontieren das Ganze, weil wir uns dann einen Zeitschritt später befinden. Indem wir unsere sofortige erwartete Belohnung addieren, erhalten wir die gesamte erwartete kumulative Belohnung für Aktion a_t. Wir wählen dann die beste der möglichen Aktionen. Gleichung 18.6 ist als *Bellmansche Gleichung* (Bellman 1957) bekannt. Auf ähnliche Weise können wir notieren:

BELLMANSCHE GLEICHUNG

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \,. \tag{18.7}$$

Sobald wir die $Q^*(s_t, a_t)$-Werte haben, können wir dann unsere Taktik π als die Ausführung von Aktion a_t^* definieren, welche den höchsten Wert von allen $Q^*(s_t, a_t)$ besitzt:

$$\pi^*(s_t) : \text{Wähle } a_t^* \text{ mit } Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t) \,. \tag{18.8}$$

Das bedeutet, wenn wir die $Q^*(s_t, a_t)$-Werte haben, dann erhalten wir durch Anwendung einer Greedy-Suche bei jedem *lokalen Schritt* die optimale Sequenz an Schritten, welche die *kumulative* Belohnung maximiert.

18.4 Modellbasiertes Lernen

Wir beginnen mit dem modellbasierten Lernen, bei dem uns die Umweltmodellparameter $p(r_{t+1}|s_t, a_t)$ und $P(s_{t+1}|s_t, a_t)$ vollkommen bekannt sind. In so einem Fall besteht kein Bedarf für Exploration und wir können direkt die optimale Wertfunktion und Taktik mittels dynamischen Programmierens bestimmen. Die optimale Wertfunktion ist eindeutig und entspricht der Lösung für die simultanen in Gleichung 18.6 gegebenen Gleichungen. Sobald uns die optimale Wertfunktion vorliegt, besteht die optimale Taktik darin, diejenige Aktion zu wählen, die den Wert im Folgezustand maximiert:

$$\pi^*(s_{t+1}) = \operatorname*{argmax}_{a_t} \Big(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s_t, a_t) V^*(s_t) \Big). \tag{18.9}$$

18.4.1 Wertiteration

```
Initialisiere V(s) mit willkürlichen Werten
Repeat
    For alle s ∈ S
        For alle a ∈ A
            Q(s,a) ← E[r|s,a] + γ Σ_{s'∈S} P(s'|s,a)V(s')
        V(s) ← max_a Q(s,a)
Until V(s) konvergieren
```

***Listing 18.1*:** Wertiterationsalgorithmus für modellbasiertes Lernen.

Um die optimale Taktik zu finden, können wir die optimale Wertfunktion nutzen, und es gibt einen iterativen Algorithmus namens *Wertiteration*, für welchen nachgewiesen wurde, dass er zu den korrekten V^*-Werten hin konvergiert. Sein Pseudocode ist im Listing 18.1 dargestellt.

WERTITERATION

Wir sagen, dass die Werte konvergiert sind, wenn die maximale Wertdifferenz zwischen zwei Iterationen kleiner als ein gewisser Schwellwert δ ist:

$$\max_{s \in \mathcal{S}} |V^{(l+1)}(s) - V^{(l)}(s)| < \delta$$

mit l als Iterationszählvariable. Weil uns lediglich die Aktionen mit dem maximalen Wert interessieren, ist es möglich, dass die Taktik zur optimalen konvergiert, selbst bevor die Werte zu ihren optimalen Werten hin konvergieren. Jede Iteration ist $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$, jedoch existiert oft nur eine kleine Zahl $k < |\mathcal{S}|$ an möglichen Folgezuständen, so dass die Komplexität sich auf $\mathcal{O}(k|\mathcal{S}||\mathcal{A}|)$ verringert.

18.4.2 Taktikiteration

Bei der Taktikiteration speichern und aktualisieren wir lieber die Taktik, statt dies indirekt über die Werte zu vollziehen. Der Pseudocode ist im Listing 18.2 gegeben. Die Idee besteht darin, mit einer Taktik anzufangen und sie wiederholt zu verbessern, bis es keine Veränderung mehr gibt. Durch Lösen der linearen Gleichungen kann die Wertfunktion berechnet werden. Wir überprüfen dann, ob wir die Taktik verbessern können, indem wir diese in unsere Betrachtungen einbeziehen. Dieser Schritt garantiert eine Verbesserung der Taktik, und wenn keine Verbesserung mehr möglich ist, dann ist gewährleistet, dass die Taktik optimal ist. Die Rechenzeit jeder Iteration dieses Algorithmus ist in $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2 + |\mathcal{S}|^3)$, was mehr ist als bei der Wertiteration, jedoch benötigt die Taktikiteration weniger Durchläufe als die Wertiteration.

18.5 Lernen mit temporaler Differenz

Ein Modell ist durch die Belohnung und die Wahrscheinlichkeitsverteilungen der Folgezustände definiert, und wie wir in Abschnitt 18.4 gesehen

Initialisiere eine Taktik π willkürlich
Repeat
  $\pi \leftarrow \pi'$
  Berechne die Werte mittels π durch
    Lösen der linearen Gleichungen
      $V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}(s')$
  Verbessere die Taktik bei jedem Zustand
    $\pi'(s) \leftarrow \underset{a}{\text{argmax}}(E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s'))$
Until $\pi = \pi'$

Listing 18.2: Algorithmus der Taktikiteration für modellbasiertes Lernen.

haben, können wir das Ganze bei Kenntnis dieser für die optimale Taktik lösen, indem wir dynamisch programmieren. Jedoch sind diese Methoden kostspielig und nur selten besitzen wir solch perfekte Kenntnis der Umgebung.

Die interessantere und realistischere Anwendung des bestärkenden Lernens findet dann statt, wenn uns kein Modell vorliegt. Dann wird eine Exploration der Umwelt erforderlich, um das Modell herauszufinden. Zuerst befassen wir uns damit, wie diese Exploration vonstatten geht, und später betrachten wir modellfreie Lernalgorithmen für deterministische und nichtdeterministische Fälle. Zwar werden wir nicht von vollkommener Kenntnis des Umweltmodells ausgehen, jedoch stellen wir die Bedingung, dass es sich um ein stationäres handelt.

Wie wir in Kürze sehen werden, nutzen wir bei der Exploration, wenn wir den Wert des Folgezustands und der Belohnung zu sehen bekommen, diese Information, um den Wert des gegenwärtigen Zustands zu aktualisieren. Diese Algorithmen nennt man Algorithmen mit *temporaler Differenz*, denn uns interessiert die Differenz zwischen unserer momentanen Schätzung des Wertes eines Zustands (oder eines Zustand-Aktions-Paares) und dem diskontierten Wert des Folgezustands und der erhaltenen Belohnung.

TEMPORALE DIFFERENZ

18.5.1 Explorationsstrategien

Für die Exploration bietet sich eine Möglichkeit in der Verwendung der ϵ-*Greedy-Suche*, bei der wir mit Wahrscheinlichkeit ϵ eine Aktion rein zufällig aus allen möglichen Aktionen wählen; das nennt man Explorieren. Mit Wahrscheinlichkeit $1 - \epsilon$ selektieren wir dabei die beste Aktion; dann spricht man vom Exploitieren. Wir wollen die Exploration nicht unendlich lang fortsetzen, sondern die Exploitation beginnen, sobald wir ausreichend Exploration vollzogen haben; darum beginnen wir mit einem hohen ϵ-Wert und senken ihn schrittweise ab. Wir müssen sicherstellen, dass unsere Taktik *weich* ist, das heißt, dass die Wahrscheinlichkeit, eine beliebige Aktion $a \in \mathcal{A}$ im Zustand $s \in \mathcal{S}$ zu wählen, größer ist als 0.

Wir können probabilistisch auswählen, indem wir die Softmax-Funktion nutzen, um Werte in Wahrscheinlichkeiten zu konvertieren,

$$P(a|s) = \frac{\exp Q(s,a)}{\sum_{b \in \mathcal{A}}^{\mathcal{A}} \exp Q(s,b)} , \tag{18.10}$$

und dann entsprechend dieser Wahrscheinlichkeiten eine Aktion zufällig wählen. Um schrittweise von Exploration zu Exploitation überzugehen, können wir eine „Temperaturvariable“ T nutzen und definieren die Wahrscheinlichkeit für die Auswahl von Aktion a als

$$P(a|s) = \frac{\exp[Q(s,a)/T]}{\sum_{b \in \mathcal{A}}^{\mathcal{A}} \exp[Q(s,b)/T]} . \tag{18.11}$$

Ist T groß, dann sind alle Wahrscheinlichkeiten gleich und es handelt sich um Exploration. Ist T klein, dann werden bessere Aktionen vorgezogen. Somit lautet die Strategie, mit einem großen Wert für T zu beginnen und ihn schrittweise zu verkleinern – eine Prozedur, die als Annealing bekannt ist – was in diesem Fall zu einem sanften zeitlichen Übergang von der Exploration zur Exploitation führt.

18.5.2 Deterministische Belohnungen und Aktionen

Beim modellfreien Lernen diskutieren wir zuerst den einfacheren deterministischen Fall, bei dem es zu jedem Zustand-Aktions-Paar eine einzelne Belohnung und möglichen Folgezustand gibt. In diesem Fall reduziert sich Gleichung 18.7 auf

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{18.12}$$

und wir nutzen dies einfach als Anweisung, $Q(s_t, a_t)$ zu aktualisieren. Wenn wir uns im Zustand s_t befinden, dann wählen wir Aktion a_t durch eine der bereits behandelten stochastischen Strategien, die eine Belohnung r_{t+1} liefert und uns in den Zustand s_{t+1} überführt. Dann aktualisieren wir den Wert der *vorangegangen* Aktion als

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) \,, \tag{18.13}$$

wobei der Zirkumflex anzeigt, dass es sich bei dem Wert um eine Schätzung handelt. $\hat{Q}(s_{t+1}, a_{t+1})$ ist ein späterer Wert und hat eine höhere Chance, korrekt zu sein. Wir diskontieren dies durch γ, addieren die sofortige Belohnung (falls vorhanden) und nutzen dies als neue Schätzung für den vorherigen $\hat{Q}(s_t, a_t)$-Wert. Hierbei spricht man von *Backup* oder *Absicherung*, da man es so betrachten kann, dass der geschätzte Wert einer Aktion im nächsten Zeitschritt herbeigenommen wird und „abgesichert“ wird, um den Schätzwert für eine gegenwärtige Aktion neu zu bewerten.

Backup

Für den Moment nehmen wir an, dass alle $\hat{Q}(s, a)$-Werte in einer Tabelle gespeichert werden; wir werden uns später damit befassen, wie wir diese Informationen kompakter speichern können, wenn $|\mathcal{S}|$ und $|\mathcal{A}|$ groß sind.

Anfänglich sind alle $\hat{Q}(s_t, a_t)$ gleich 0 und werden mit der Zeit als Ergebnis von Versuchsepisoden aktualisiert. Nehmen wir an, uns liegt eine Sequenz an Zügen vor, und bei jedem Zug nutzen wir Gleichung 18.13, um die Schätzung des Q-Wertes des vorangegangen Zustand-Aktions-Paares zu aktualisieren, und zwar unter Verwendung des Q-Wertes des momentanen Zustand-Aktions-Paares. In den Zwischenzuständen sind alle Belohnungen und somit alle Werte gleich 0, so dass keine Aktualisierung vollzogen wird. Wenn wir den Zielzustand erreichen, erhalten wir die Belohnung

r und können dann den Q-Wert des vorangegangen Zustand-Aktions-Paares als γr aktualisieren. Wie für das vorherige Zustand-Aktions-Paar ist die sofortige Belohnung gleich 0 und der Beitrag des nächsten Zustand-Aktions-Paares wird durch γ diskontiert, weil wir einen Schritt weitergegangen sind. Wenn wir dann in einer weiteren Episode diesen Zustand erreichen, können wir den entsprechend vorangegangenen als $\gamma^2 r$ aktualisieren, und so weiter. Auf diese Weise wird diese Information nach vielen Episoden auf früheren Zustand-Aktions-Paaren abgesichert. Die Q-Werte nehmen bis zu ihren optimalen Werten zu, wenn wir Pfade mit höherer kumulativer Belohnung finden, beispielsweise kürzere Pfade, jedoch nehmen sie niemals ab (siehe Abbildung 18.2).

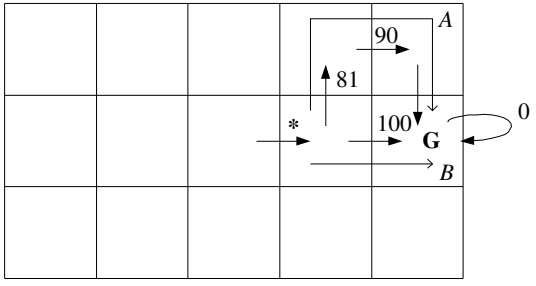


***Abb. 18.2*:** Ein Beispiel, um zu zeigen, dass die Q-Werte zunehmen aber nie abnehmen. Dies ist eine deterministische Rasterwelt, bei der G der Zielzustand mit Belohnung 100 ist, alle anderen Belohnungen gleich 0 sind und $\gamma = 0{,}9$. Betrachten wir den Q-Wert des durch einen Stern markierten Übergangs und befassen wir uns einfach mal nur mit den Pfaden A und B. Sagen wir, Pfad A wird vor Pfad B gesehen, dann erhalten wir $\gamma \max(0{,}81) = 72{,}9$. Wenn B danach gesehen wird, dann wird ein kürzerer Pfad gefunden und der Q-Wert wird $\gamma \max(100{,}81) = 90$. Ist B vor A bekannt, dann ist der Q-Wert $\gamma \max(100{,}0) = 90$. Wenn dann B gesehen wird, verändert er sich nicht, da $\gamma \max(100{,}81) = 90$.

Man beachte, dass wir hier die Belohnung oder Folgezustandsfunktionen nicht kennen. Sie sind ein Teil der Umwelt und es ist, als ob wir sie während der Exploration erfragen. Wir modellieren sie auch nicht, obwohl dies eine weitere Möglichkeit wäre. Wir akzeptieren sie einfach als gegeben und lernen direkt die optimale Taktik durch die geschätzte Wertfunktion.

18.5.3 Nichtdeterministische Belohnungen und Aktionen

Sind die Belohnungen und das Ergebnis von Aktionen nicht deterministisch, dann haben wir eine Wahrscheinlichkeitsverteilung für die Belohnung $p(r_{t+1}|s_t, a_t)$, aus der Belohnungen stichprobenartig entnommen werden, und es existiert eine Wahrscheinlichkeitsverteilung für den Folgezustand $P(s_{t+1}|s_t, a_t)$. Diese helfen uns, die Ungewissheit im System zu

modellieren, welche sich möglicherweise aus Kräften der Umwelt ergibt, die sich unserem Einfluss entziehen: zum Beispiel unser Gegner beim Schach, die Würfel beim Backgammon oder unser Mangel an Kenntnis des Systems. Beispielsweise könnten wir es mit einem imperfekten Roboter zu tun haben, dem es manchmal nicht gelingt, sich in die beabsichtigte Richtung zu bewegen und der vom Kurs abweicht oder kürzer bzw. weiter voranschreitet als erwartet.

In so einem Fall erhalten wir

$$Q(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) . \tag{18.14}$$

Dann ist eine direkte Zuordnung nicht möglich, da wir für denselben Zustand und dieselbe Aktion unter Umständen verschiedene Belohnungen erhalten oder in unterschiedliche Folgezustände übergehen. Stattdessen verwenden wir einen Durchschnitt. Dabei spricht man vom *Q-Lernalgorithmus*:

Q-Lernalgorithmus

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) . \tag{18.15}$$

Wir stellen uns die Werte für $r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$ als eine Stichprobe aus Instanzen für jedes (s_t, a_t)-Paar vor und wir wollen, dass $\hat{Q}(s_t, a_t)$ zu seinem Mittelwert konvergiert. Wie üblich wird η zum Zwecke der Konvergenz schrittweise in der Zeit verringert und es ist bewiesen, dass dieser Algorithmus zu den optimalen Q^*-Werten konvergiert (Watkins und Dayan 1992). Der Pseudocode des Q-Lernalgorithmus ist im Listing 18.3 gegeben.

```
Initialisiere alle Q(s, a) willkürlich
For alle Episoden
    Initialisiere s
    Repeat
        Wähle a mittels der Taktik aus Q, z.B. ϵ-Greedy
        Führe Aktion a aus, beobachte r und s′
        Aktualisiere Q(s, a):
            Q(s, a) ← Q(s, a) + η(r + γ max_a′ Q(s′, a′) − Q(s, a))
        s ← s′
    Until s ist im finalen Zustand
```

***Listing 18.3*:** Q-Lernalgorithmus, ein Algorithmus mit temporaler Differenz, der sich abseits der Taktik bewegt.

Wir können uns Gleichung 18.15 auch als eine Reduktion der Differenz zwischen dem aktuellen Q-Wert und der abgesicherten Schätzung aus einem Zeitschritt später vorstellen. Solche Algorithmen bezeichnet man als Algorithmen mit *temporaler Differenz* (TD-Algorithmen; siehe auch Sutton 1988).

temporale Differenz

```
Initialisiere alle Q(s, a) willkürlich
For alle Episoden
    Initialisiere s
    Wähle a mittels der Taktik aus Q, z.B. ϵ-Greedy
    Repeat
        Führe Aktion a aus, beobachte r und s'
        Wähle a' mittels der Taktik aus Q, z.B. ϵ-greedy
        Aktualisiere Q(s, a):
            Q(s, a) ← Q(s, a) + η(r + γQ(s', a') − Q(s, a))
        s ← s', a ← a'
    Until s ist im finalen Zustand
```

***Listing 18.4*:** Sarsa-Algorithmus, eine taktikbasierte Version des Q-Lernalgorithmus.

taktikfreie Methode
taktikbasierte Methode
Sarsa-Algorithmus

Es handelt sich um eine *taktikfreie* Methode, da der Wert der besten Folgeaktion ohne Einsatz einer Taktik genutzt wird. Dagegen nutzt man bei einer *taktikbasierten* Methode die Taktik, um auch die Folgeaktion zu bestimmen. Die taktikbasierte Version des Q-Lernalgorithmus ist der *Sarsa-Algorithmus*, dessen Pseudocode im Listing 18.4 dargestellt ist. Wir erkennen, dass statt alle möglichen Folgeaktionen a' zu suchen und die beste zu wählen, der taktikbasierte Sarsa-Algorithmus die von Q-Werten abgeleitete Taktik nutzt, um eine Folgeaktion a' zu wählen und deren Q-Wert für die Berechnung der temporalen Differenz einsetzt. Taktikbasierte Methoden schätzen den Wert einer Taktik, während sie ihn nutzen, um Aktionen durchzuführen. Bei taktikfreien Methoden sind diese separat und die zur Verhaltensgenerierung eingesetzte Taktik, auch als *Verhaltenstaktik* bekannt, kann sich tatsächlich von der Taktik unterscheiden, die evaluiert und verbessert wird, die sogenannte *Schätzungstaktik*.

Sarsa konvergiert mit der Wahrscheinlichkeit 1 zur optimalen Taktik und damit zu optimalen Werten der Zustand-Aktions-Paare, falls eine *GLIE-Taktik* für die Auswahl von Aktionen eingesetzt wird. Eine GLIE-Taktik (engl. *greedy in the limit with infinite exploration*) liegt vor, wenn (1) alle Zustand-Aktions-Paare unendlich oft aufgesucht werden und (2) die Taktik im Endlichen zur Greedy-Taktik konvergiert (was beispielsweise mit Hilfe von ϵ-Greedy-Taktiken durch Festlegung von $\epsilon = 1/t$ arrangiert werden kann).

TD-Lernen

Die gleiche Vorstellung von temporaler Differenz kann auch verwendet werden, um $V(s)$-Werte statt der $Q(s, a)$ zu lernen. Beim *TD-Lernen* (Sutton 1988) wird die folgende Aktualisierungsregel verwendet, um einen Zustandswert zu aktualisieren:

$$V(s_t) \leftarrow V(s_t) + \eta[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \,. \tag{18.16}$$

Erneut handelt es sich um die Deltaregel, bei der $r_{t+1} + \gamma V(s_{t+1})$ die bessere, spätere Vorhersage und $V(s_t)$ die momentane Schätzung ist.

Ihre Differenz ist die temporale Differenz und die Aktualisierung wird vorgenommen, um diesen Unterschied zu verringern. Der Aktualisierungsfaktor η wird schrittweise verringert und TD konvergiert garantiert zur optimalen Wertfunktion $V^*(s)$.

18.5.4 Eignungsprotokolle

Die eben beschriebenen Algorithmen sind Einschrittprozeduren, das heißt, die temporale Differenz wird genutzt, um nur den vorangegangenen Wert (des Zustands oder des Zustand-Aktions-Paares) zu aktualisieren.

EIGNUNGSPROTOKOLL

Ein *Eignungsprotokoll* (engl. *eligibility trace*) enthält Aufzeichnungen zur Häufigkeit der Paare in der Vergangenheit und ermöglicht uns, die Zuweisung einer erhaltenen Belohnung zur verantwortlichen Aktion auch in temporaler Form zu implementieren; dadurch werden wir in die Lage versetzt, auch die Werte von vorher aufgetretenen Paaren zu aktualisieren. Wir betrachten, wie dies mit Sarsa durchgeführt wird, um Q-Werte zu erlernen; die Adaption dessen, um V-Werte zu erlernen, funktioniert analog dazu.

Um das Eignungsprotokoll zu speichern, benötigen wir eine zusätzliche Speichervariable, $e(s,a)$, die mit jedem Zustand-Aktions-Paar assoziiert und mit 0 initiiert wird. Wenn das Zustand-Aktions-Paar (s,a) auftritt, sprich, wenn wir Aktion a im Zustand s ausführen, dann wird dessen Eignung auf 1 gesetzt; die Eignungen aller anderen Zustand-Aktions-Paare werden mit $\gamma\lambda$ multipliziert. $0 \leq \lambda \leq 1$ ist der Parameter des Verfalls des Eignungsprotokolls.

$$e_t(s,a) = \begin{cases} 1 & \text{falls } s = s_t \text{ und } a = a_t\,, \\ \gamma\lambda e_{t-1}(s,a) & \text{andernfalls}\,. \end{cases} \tag{18.17}$$

Wenn ein Zustand-Aktions-Paar noch nie eingetreten ist, dann bleibt dessen Eignung gleich 0; ist dies jedoch schon vorgekommen, dann fällt dessen Eignung mit voranschreitender Zeit und wenn andere Zustand-Aktions-Paare eintreten von 1 ab; außerdem besteht eine Abhängigkeit von den Werten für γ und λ (siehe Abbildung 18.3).

Wir erinnern uns, dass der temporale Fehler zum Zeitpunkt t bei Sarsa wie folgt lautet

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\,. \tag{18.18}$$

Bei Sarsa mit Eignungsprotokoll, bezeichnet mit Sarsa(λ), werden *alle* Zustand-Aktions-Paare aktualisiert mit

$$Q(s,a) \leftarrow Q(s,a) + \eta\delta_t e_t(s,a),\ \forall s,a\,. \tag{18.19}$$

Dadurch werden alle in Frage kommenden Zustand-Aktions-Paare aktualisiert, wobei die Aktualisierung davon abhängt, wie weit in der Vergangenheit das Auftreten der Paare zurückliegt. Der Wert für λ definiert die

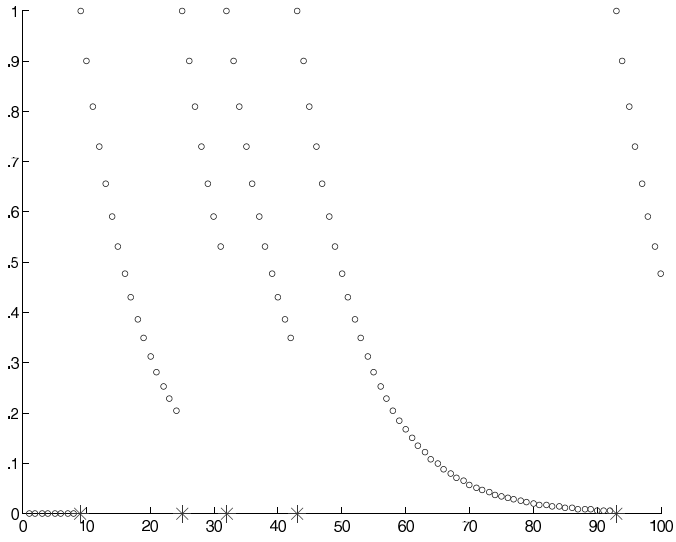


***Abb. 18.3*:** Beispiel eines Eignungsprotokolls für einen Wert. Das Auftreten der Zustand-Aktions-Paare wird durch einen Stern markiert.

temporale Zuordnung einer erhaltenen Belohnung zu einer Aktion: falls $\lambda = 0$, dann wird nur eine Einschrittaktualisierung durchgeführt. Die Algorithmen, die wir in Abschnitt 18.5.3 diskutiert haben, fallen in diese Kategorie und werden aus diesem Grund mit $Q(0)$, Sarsa(0) oder TD(0) bezeichnet. Mit Annäherung von λ an 1 werden mehr der vorangegangenen Schritte einbezogen. Falls $\lambda = 1$, dann werden alle vorherigen Schritte aktualisiert und der Wert für die ihnen zugeordnete Belohnung verringert sich nur um γ je Schritt. Bei einer online Aktualisierung werden alle in Frage kommenden Werte nach jedem Schritt sofort aktualisiert; bei der offline Aktualisierung werden die Aktualisierungen angesammelt und eine einzelne Aktualisierung wird am Ende einer Episode durchgeführt. Die online Aktualisierung dauert länger, konvergiert jedoch schneller. Der Pseudocode für *Sarsa(λ)* findet sich im Listing 18.5. Die $Q(\lambda)$- und TD(λ)-Algorithmen können auf vergleichbare Weise abgeleitet werden (Sutton und Barto 1998). SARSA(λ)

18.6 Generalisierung

Bis jetzt sind wir davon ausgegangen, dass die $Q(s, a)$-Werte (oder die $V(s)$, falls wir Werte von Zuständen schätzen) in einer Lookup-Tabelle abgespeichert werden und die von uns bisher betrachteten Algorithmen werden als *tabulare* Algorithmen bezeichnet. Es existieren eine Menge Probleme mit diesem Ansatz: (1.) Ist die Zahl an Zuständen und an Aktionen groß, dann kann auch die Größe der Tabelle enorme Dimensionen

```
Initialisiere alle Q(s, a) willkürlich, e(s, a) ← 0, ∀s, a
For alle Episoden
    Initialisiere s
    Wähle a mittels der Taktik aus Q, z.B. ϵ-Greedy
    Repeat
        Führe Aktion a aus, beobachte r und s′
        Wähle a′ mittels der Taktik aus Q, z.B. ϵ-Greedy
        δ ← r + γQ(s′, a′) − Q(s, a)
        e(s, a) ← 1
        For alle s, a:
            Q(s, a) ← Q(s, a) + ηδe(s, a)
            e(s, a) ← γλe(s, a)
        s ← s′, a ← a′
    Until s ist im finalen Zustand
```

Listing 18.5: Sarsa(λ)-Algorithmus.

annehmen. (2.) Zustände und Aktionen können kontinuierlich sein, beispielsweise das Drehen eines Lenkrads um einen bestimmten Winkel; um eine Tabelle nutzen zu können, bedarf es ihrer Diskretisierung, wodurch Fehler entstehen können. (3.) Wenn der Suchraum groß ist, dann sind unter Umständen zu viele Episoden nötig, um alle Einträge der Tabelle mit akzeptabler Genauigkeit zu füllen.

Statt die Q-Werte so zu speichern, wie sie sind, können wir das Ganze als Regressionsproblem betrachten. Es handelt sich um eine Aufgabenstellung des überwachten Lernens, bei der wir einen Regressor $Q(s, a|\boldsymbol{\theta})$ definieren, der s und a als Eingabe nutzt und durch einen Vektor von Parametern $\boldsymbol{\theta}$ parameterisiert wird, um Q-Werte zu lernen. Zum Beispiel kann dies ein künstliches neuronales Netz sein mit s und a als Eingaben, einer Ausgabe und $\boldsymbol{\theta}$ als Verbindungsgewichtungen.

Ein gutes Näherungsmodell weist die üblichen Vorteile auf und löst die soeben besprochenen Probleme. Eine gute Approximation kann mit einem einfachen Modell erzielt werden, ohne explizit die Trainingsinstanzen abzuspeichern; sie kann kontinuierliche Eingaben nutzen und erlaubt die Generalisierung: wenn wir wissen, dass ähnliche (s, a)-Paare ähnliche Q-Werte besitzen, dann können wir anhand vergangener Fälle generalisieren und zu vorteilhaften $Q(s, a)$-Werten gelangen, selbst wenn jenes Zustand-Aktions-Paar noch nie zuvor eingetreten ist.

Um in der Lage zu sein, den Regressor zu trainieren, benötigen wir einen Trainingsdatensatz. Im Fall von Sarsa(0) haben wir bereits gesehen, dass wir $Q(s_t, a_t)$ so nah wie möglich an $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ heranbringen wollen. Somit können wir eine Menge an Trainingsstichproben formen, bei denen die Eingabe das Zustand-Aktions-Paar (s_t, a_t) und die geforderte

Ausgabe $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ ist. Wir können den quadratischen Fehler notieren als

$$E^t(\boldsymbol{\theta}) = [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]^2 \,. \tag{18.20}$$

Trainingsdatensätze können auf vergleichbare Weise für $Q(0)$ und TD(0) definiert werden, wobei wir bei Letzterem $V(s)$ lernen und die geforderte Ausgabe gleich $r_{t+1} - \gamma V(s_{t+1})$ ist. Sobald solch ein Trainingsdatensatz bereit ist, können wir jeden beliebigen Algorithmus des überwachten Lernens anwenden, um den Trainingsdatensatz zu erlernen.

Nutzen wir eine Gradientenabstiegsmethode, wie beim Training neuronaler Netze, dann wird der Parametervektor aktualisiert als

$$\Delta\boldsymbol{\theta} = \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t) \,. \tag{18.21}$$

Es handelt sich um eine Einschrittaktualisierung. Im Falle von Sarsa(λ) wird das Eignungsprotokoll ebenfalls einbezogen:

$$\Delta\boldsymbol{\theta} = \eta\delta_t \boldsymbol{e}_t \,, \tag{18.22}$$

wobei der Fehler der temporalen Differenz sich als

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

ergibt und der Vektor der Eignungen der Parameter aktualisiert wird mittels

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t) \tag{18.23}$$

mit allen $\boldsymbol{e}_0$ gleich 0. Im Falle eines tabularen Algorithmus werden die Eignungen für die Zustand-Aktions-Paare gespeichert, da diese die Parameter darstellen (gespeichert in Form einer Tabelle). Im Falle eines Schätzers wird die Eignung mit den Parametern des Schätzers assoziiert. Wir bemerken auch, dass das Ganze der Momentumsmethode für die Stabilisierung der Rückpropagierung (Abschnitt 11.8.1) ähnelt. Der Unterschied besteht darin, dass im Falle von Momentum vorangegangene Gewichtsänderungen erinnert werden, wohingegen es hier vorangegangene Gradientenvektoren sind. In Abhängigkeit vom für $Q(s_t, a_t)$ verwendeten Modell, beispielsweise einem neuronalen Netz, setzen wir den Gradientenvektor in Gleichung 18.23 ein.

Theoretisch kann jede beliebige Regressionsmethode verwendet werden, um die Q-Funktion zu trainieren, doch die spezifische Aufgabenstellung bringt eine Reihe von Anforderungen mit sich: Erstens sollte die Generalisierung möglich sein, das heißt, wir müssen wirklich gewährleisten, dass ähnliche Zustände und Aktionen auch ähnliche Q-Werte aufweisen. Das bedingt auch eine angemessene Codierung von s und a wie bei jeder Applikation, um die Ähnlichkeiten zum Vorschein zu bringen. Zweitens

stellen Aktualisierungen des bestärkenden Lernens eine Instanz nach der anderen bereit und nicht den gesamten Trainingsdatensatz. Der Lernalgorithmus sollte also in der Lage sein, individuelle Aktualisierungen einzubringen, um die neue Instanz zu lernen, ohne dabei zu vergessen, was bereits vorher gelernt wurde. Zum Beispiel kann ein mehrlagiges Perzeptron, welches Rückpropagierung anwendet, nur dann mit einer einzelnen Instanz trainiert werden, wenn nur eine kleine Lernrate verwendet wird. Alternativ können derlei Instanzen gesammelt werden, um einen Trainingsdatensatz zu bilden und dann insgesamt erlernt werden, doch verlangsamt sich dadurch der Lernprozess, da keinerlei Lernen stattfindet, während eine ausreichend große Stichprobe angesammelt wird.

Aus diesen Gründen erscheint es eine gute Idee, lokale Lerner zu verwenden, um die Q-Werte zu lernen. Bei solchen Methoden, zum Beispiel radialen Basisfunktionen, werden Informationen lokalisiert, und wenn eine neue Instanz erlernt ist, wird nur ein lokaler Teil des Lerners aktualisiert, ohne möglicherweise die Informationen in einem anderen Teil zu korrumpieren. Dieselben Anforderungen gelten, wenn wir die Zustandswerte als $V(s_t|\boldsymbol{\theta})$ schätzen.

18.7 Teilweise beobachtbare Zustände

In gewissen Anwendungen kennt der Agent den Zustand nicht genau. Er ist mit Sensoren ausgestattet, die eine *Beobachtung* liefern, mit deren Hilfe der Agent den Zustand schätzen sollte. Nehmen wir an, wir haben einen Roboter, der sich in einem Raum bewegt. Der Roboter kennt möglicherweise seine exakte Position im Raum nicht, noch weiß er, was sich sonst noch im Raum befindet. Der Roboter kann mit einer Kamera versehen sein, mit der sensorische Beobachtungen aufgezeichnet werden. Dadurch wird dem Roboter nicht exakt sein Zustand mitgeteilt, aber er erhält einige Hinweise auf seinen wahrscheinlichen Zustand. Zum Beispiel könnte der Roboter nur wissen, dass sich zu seiner Rechten eine Wand befindet.

TEILWEISE BEOBACHTBARER MDP

Dieses Szenario entspricht einem Markov-Entscheidungsprozess, außer dass nach der Durchführung von Aktion a_t der neue Zustand s_{t+1} unbekannt ist und wir stattdessen nur über eine Beobachtung o_{t+1} verfügen, die einer stochastischen Funktion von s_t und a_t entspricht: $p(o_{t+1}|s_t, a_t)$. Dies wird als *teilweise beobachtbarer Markov-Entscheidungsprozess* (engl. *partially observable Markov decision process*, POMDP) bezeichnet. Falls $o_{t+1} = s_{t+1}$, dann reduziert sich ein POMDP auf einen MDP. Das ist genau wie die Unterscheidung zwischen beobachtbaren und Hidden-Markov-Modellen und auch die Lösung ist ähnlich; das heißt, wir müssen anhand der Beobachtung den Zustand ableiten (oder besser gesagt, eine Wahrscheinlichkeitsverteilung für die Zustände) und dann agieren. Wenn der Agent glaubt, dass er sich im Zustand s_1 mit Wahrscheinlichkeit 0,4 und im Zustand s_2 mit Wahrscheinlichkeit 0,6 befindet, dann ist der Wert

einer beliebigen Aktion gleich 0,4 Mal der Wert der Aktion in s_1 plus 0,6 Mal der Wert der Aktion in s_2.

Die Markov-Eigenschaft ist für Beobachtungen nicht haltbar: die Folgezustandsbeobachtung hängt nicht nur von der momentanen Aktion und Beobachtung ab. Im Falle von begrenzten Beobachtungen können zwei Zustände als identisch erscheinen, die aber unterschiedlich sind, und wenn diese Zustände verschiedene Aktionen erfordern, dann kann das zu einem Verlust an Leistungsfähigkeit führen, wie sie durch die kumulative Belohnung gemessen wird. Der Agent sollte auf irgendeine Weise den vergangenen Ablauf in seine gegenwärtige eindeutige Zustandsschätzung einbringen. Diese vergangenen Beobachtungen können auch einbezogen werden, indem ein vergangenes Beobachtungsfenster als Eingabe für die Taktik genutzt wird, oder indem man ein rekurrentes neuronales Netz (Abschnitt 11.12.2) einsetzt, um den Zustand zu erhalten, ohne vergangene Beobachtungen zu vergessen.

Zu jedem beliebigen Zeitpunkt kann der Agent den wahrscheinlichsten Zustand berechnen und eine entsprechende Aktion ausführen. Oder er kann eine Aktion initiieren, um Informationen zu sammeln, um die Ungewissheit zu reduzieren, beispielsweise nach einem Wahrzeichen suchen oder anhalten, um nach dem Weg zu fragen. Dies impliziert die Wichtigkeit des *Informationswerts*, und tatsächlich kann ein POMDP als *dynamisches* Einflussdiagramm (Abschnitt 14.8) modelliert werden. Der Agent wählt Aktionen aus, anhand der Menge an Informationen, die sie bereitstellen, anhand der Höhe der Belohnung, die sie produzieren und anhand dessen, wie sie den Zustand der Umwelt verändern.

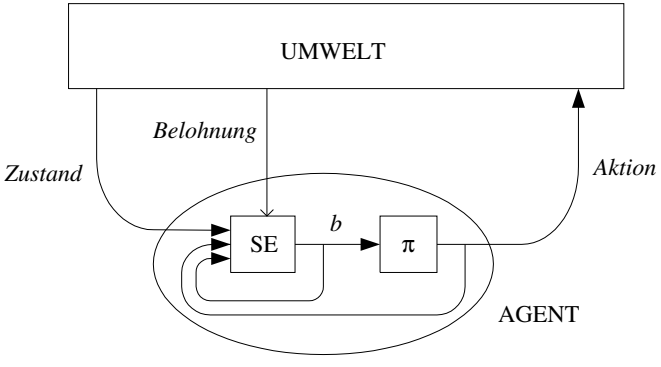


Abb. 18.4: Im Falle einer teilweise beobachtbaren Umgebung hat der Agent einen Zustandsschätzer (engl. *state estimator*, SE) der einen internen Belief-Zustand b aufrechterhält, und die Taktik π generiert Aktionen basierend auf den Belief-Zuständen.

Um den Prozess in Markov-Form beizubehalten, besitzt der Agent einen internen *Belief-Zustand* b_t (vom engl. *belief* für „Glauben“), der seine Erfahrungen zusammenfasst (siehe Abbildung 18.4). Der Agent besitzt

Belief-Zustand

einen *Zustandsschätzer*, welcher den Belief-Zustand b_{t+1} anhand der letzten Aktion a_t, anhand der momentanen Beobachtung o_{t+1} und anhand dem vorherigen Belief-Zustand b_t aktualisiert. Es existiert eine Taktik π, welche die Folgeaktion a_{t+1} gemäß diesem Belief-Zustand generiert, im Gegensatz zum realen Zustand, der uns in einer komplett beobachtbaren Umwelt zugänglich wäre. Der Belief-Zustand ist eine Wahrscheinlichkeitsverteilung über Zustände der Umwelt bei gegebenem anfänglichen Belief-Zustand (bevor wir irgendwelche Aktionen ausgeführt haben) und der Vorgeschichte von Beobachtungen und Aktionen des Agenten (ohne dass Informationen ausgelassen werden, welche die Leistung des Agenten verbessern könnten). Das Q-Lernen involviert in so einem Fall die Werte der Paare von Belief-Zustand und Aktion statt der eigentlichen Zustand-Aktions-Paare:

$$Q(b_t, a_t) = E[r_{t+1}] + \gamma \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) V(b_{t+1}) \,. \tag{18.24}$$

18.7.1 Beispiel: Das Tigerproblem

Wir wollen nun ein Beispiel untersuchen, bei dem es sich um eine leicht abgewandelte Version des Tigerexperiments handelt, das in Kaelbling, Littman und Cassandra 1998 diskutiert wird und das wie in Thrun, Burgard und Fox 2005 modifiziert wurde. Angenommen, wir stehen vor zwei Türen – die eine zu unserer Linken, die andere zu unserer Rechten –, die jeweils in ein Zimmer führen. Hinter einer der beiden Türen hockt ein Tiger (wir wissen nicht hinter welcher) und hinter der anderen liegt ein Schatz. Falls wir die Tür öffnen, hinter der sich der Tiger verbirgt, erhalten wir eine sehr hohe negative Belohnung, und falls wir die Tür zur Schatzkammer öffnen, erhalten wir eine gewisse positive Belohnung. Der verborgene Zustand z_L ist der Aufenthaltsort des Tigers. Bezeichnen wir mit p die Wahrscheinlichkeit, dass sich der Tiger im linken Zimmer befindet. Die Wahrscheinlichkeit, dass er rechts ist, ist dann $1 - p$:

$$p \equiv P(z_L = 1) \,.$$

Die beiden Aktionen, d. h. das Öffnen der linken bzw. der rechten Tür, bezeichnen wir mit a_L und a_R. Die Belohnung sind:

$r(A, Z)$	Tiger links	Tiger rechts
links öffnen	-100	$+80$
rechts öffnen	$+90$	-100

Berechnen wir nun die erwarteten Belohnungen für die beiden Aktionen. Es gibt keine zukünftigen Belohnungen, da das Experiment endet, sobald wir eine der beiden Türen geöffnet haben.

$$R(a_L) = r(a_L, z_L)\, P(z_L) + r(a_L, z_R)\, P(z_R) = -100p + 80(1 - p)$$
$$R(a_R) = r(a_R, z_L)\, P(z_L) + r(a_R, z_R)\, P(z_R) = 90p - 100(1 - p)$$

Bei diesen Belohnungen und wenn p nahe 1 ist, d. h., wenn wir glauben, dass es eine große Chance gibt, dass sich der Tiger links befindet, ist das Öffnen der rechten Tür die richtige Aktion, und entsprechend ist es für p nahe 0 besser, die linke Tür zu öffnen.

Die beiden schneiden sich, wenn p ungefähr 0,5 ist, und dort ist die erwartete Belohnung ungefähr -10. Die Tatsache, dass die erwartete Belohnung negativ ist, wenn p ungefähr 0,5 ist (wenn wir also in Ungewissheit sind) zeigt, wie wichtig es ist, Informationen zu sammeln. Wenn wir Sensoren anbringen könnten, um die Ungewissheit zu verringern – d. h. p entweder in die Nähe von 0 oder in die Nähe von 1 zu verschieben –, dann können wir Aktionen mit hohen positiven Belohnungen ausführen. Die Aktion des Abtastens durch den Sensor, a_S, kann eine kleine negative Belohnung $R(a_S) = -1$ haben, was man als Kosten für das Abtasten ansehen kann oder auch als Diskontierung der künftigen Belohnung um $\gamma < 1$, da wir die eigentliche Aktion (Öffnen einer Tür) verschieben.

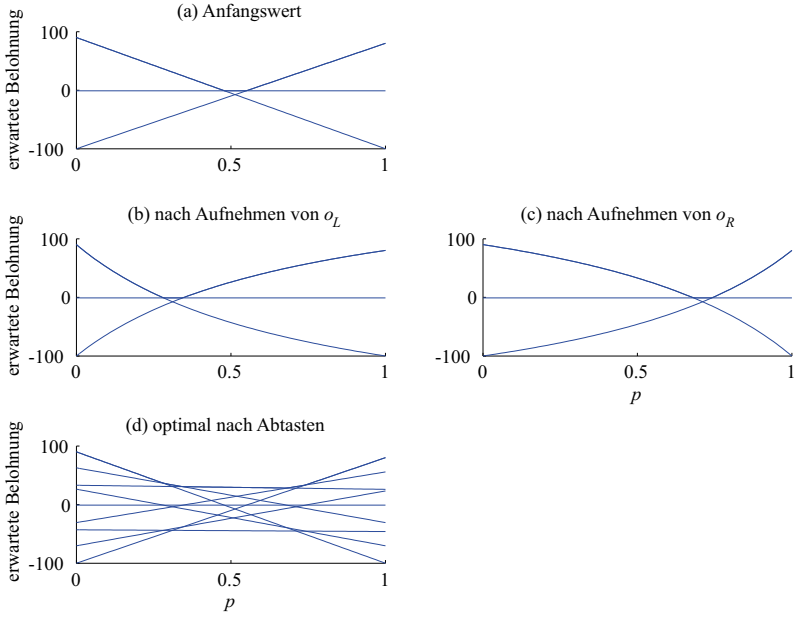


Abb. 18.5: Erwartete Belohnungen und der Effekt des Abtastens.

Für einen solchen Fall sind die erwarteten Belohnungen und der Wert der besten Aktion in Abbildung 18.5a dargestellt:

$$V = \max(a_L, a_R, a_S) \, .$$

Nehmen wir an, wir verwenden als Sensoren Mikrofone, um herauszufinden, ob sich der Tiger hinter der linken oder rechten Tür befindet.

Doch unsere Sensoren sind unzuverlässig, so dass eine gewisse Unsicherheit bleibt. Sagen wir, die Anwesenheit des Tigers kann mit einer Wahrscheinlichkeit von 0,7 detektiert werden:

$$P(o_L|z_L) = 0{,}7 \qquad (o_L|z_R) = 0{,}3$$
$$P(o_R|z_L) = 0{,}3 \qquad (o_R|z_R) = 0{,}7$$

Wenn wir o_L aufnehmen, ändert sich unsere Vermutung, wo sich der Tiger befindet:

$$p' = P(z_L|o_L) = \frac{P(o_L|z_L)\,P(z_L)}{p(o_L)} = \frac{0{,}7p}{0{,}7p + 0{,}3(1-p)}\,.$$

Der Effekt ist in Abbildung 18.5b gezeigt, in der $R(a_L|o_L)$ skizziert ist. Das Aufnehmen von o_L macht das Öffnen der rechten Tür für einen breiteren Bereich zu der besseren Aktion. Je bessere Sensoren wir verwenden (d. h., wenn die Wahrscheinlichkeit für eine korrekte Aufnahme sich von 0,7 in Richtung 1 bewegt), umso größer wird dieser Bereich (Übung 9). Wenn wir hingegen o_R aufnehmen, erhöht das entsprechend die Chancen, die linke Tür zu öffnen, was in Abbildung 18.5c dargestellt ist. Im Übrigen verringert sich durch das Abtasten auch der Bereich, in dem weiteres Abtasten nötig ist.

Die erwarteten Belohnungen für die Aktionen sind in diesem Fall

$$\begin{aligned}
R(a_L|o_L) &= r(a_L, z_L)\,P(z_L|o_L) + r(a_L, z_R)\,P(z_R|o_L) \\
&= -100p' + 80(1-p') \\
&= -100 \cdot \frac{0{,}7 \cdot p}{p(o_L)} + 80 \cdot \frac{0{,}3 \cdot (1-p)}{p(o_L)}\,, \\
R(a_R|o_L) &= r(a_R, z_L)\,P(z_L|o_L) + r(a_R, z_R)\,P(z_R|o_L) \\
&= 90p' - 100(1-p') \\
&= 90 \cdot \frac{0{,}7 \cdot p}{p(o_L)} - 100 \cdot \frac{0{,}3 \cdot (1-p)}{p(o_L)}\,, \\
R(a_S|o_L) &= -1\,.
\end{aligned}$$

Die beste Aktion ist in diesem Fall das Maximum dieser drei. Entsprechend werden die erwarteten Belohnungen, wenn wir o_R aufnehmen, zu

$$\begin{aligned}
R(a_L|o_R) &= r(a_L, z_L)\,P(z_L|o_R) + r(a_L, z_R)\,P(z_R|o_R) \\
&= -100 \cdot \frac{0{,}3 \cdot p}{p(o_R)} + 80 \cdot \frac{0{,}7 \cdot (1-p)}{p(o_R)}\,, \\
R(a_R|o_R) &= r(a_R, z_L)\,P(z_L|o_R) + r(a_R, z_R)\,P(z_R|o_R) \\
&= 90 \cdot \frac{0{,}3 \cdot p}{p(o_R)} - 100 \cdot \frac{0{,}7 \cdot (1-p)}{p(o_R)}\,, \\
R(a_S|o_R) &= -1\,.
\end{aligned}$$

Um die erwartete Belohnung zu berechnen, müssen wir über beide Sensormesswerte, gewichtet nach ihren Wahrscheinlichkeiten, mitteln:

$$\begin{aligned}
V' &= \sum_j \left[\max_i R(a_i|a_j)\right] P(O_j) \\
&= \max\big(R(a_L|o_L), R(a_R|o_L), R(a_S|o_L)\big)\, P(o_L) \\
&\quad + \max\big(R(a_L|o_R), R(a_R|o_R), R(a_S|o_R)\big)\, P(o_R) \\
&= \max\big(-70p + 24(1-p), 63p - 30(1-p), -0{,}7p - 0{,}3(1-p)\big) \\
&\quad + \max\big(-30p + 56(1-p), 27p - 70(1-p), -0{,}3p - 0{,}7(1-p)\big) \\
&= \max \begin{pmatrix} -100p & +80(1-p) \\ -43p & -46(1-p) \\ 33p & +26(1-p) \\ 90p & -100(1-p) \end{pmatrix} . \qquad (18.25)
\end{aligned}$$

Wenn wir mit $P(o_L)$ multiplizieren, erhalten wir Funktionen, die linear in p sind. Diese fünf Zeilen und die stückweise Funktion, die ihrem Maximum entspricht, sind in Abbildung 18.5d zu sehen. Man beachte, dass die Linie $-40p - 5(1-p)$ ebenso wie diejenigen, die a_S enthalten, für alle Werte von p unter den anderen liegen und daher mit Sicherheit verworfen werden können. Die Tatsache, dass Abbildung 18.5d besser ist als Abbildung 18.5a beweist den *Wert der Information.*

Wert der Information

Was wir hier berechnen, ist der Wert der besten Aktion, wenn wir a_S gewählt hätten. Beispielsweise entspricht die erste Zeile der Wahl von a_L, nachdem a_S gewählt wurde. Um also die beste Entscheidung bei einer Episode der Länge zwei zu finden, müssen wir dies aktualisieren, indem wir 1, d. h. die negative Belohnung von a_S, subtrahieren. Wir erhalten die erwartete Belohnung für die Aktion des Abtastens. Äquivalent dazu können wir dies als Warten interpretieren, was eine unmittelbare Belohnung von 0 ergibt, aber die zukünftige Belohnung um $\gamma < 1$ diskontiert. Außerdem haben wir die beiden ursprünglichen Aktionen a_L und a_R, und wir wählen die beste von diesen dreien.

Wir wollen das Problem nun noch etwas interessanter machen, wobei wir dem Beispiel aus Thrun, Burgard und Fox 2005 folgen. Dazu nehmen wir an, dass es eine Tür zwischen den beiden Zimmern gibt und der Tiger, ohne dass wir es sehen können, sich von dem einen Zimmer in das andere bewegen kann. Außerdem wollen wir annehmen, dass es sich um einen ziemlich rastlosen Tiger handelt, der mit der Wahrscheinlichkeit 0,2 im selben Zimmer bleibt und mit der Wahrscheinlichkeit 0,8 in das andere Zimmer geht. Daraus folgt, dass wir unser p wie folgt aktualisieren müssen:

$$p' = 0{,}2p + 0{,}8(1-p) .$$

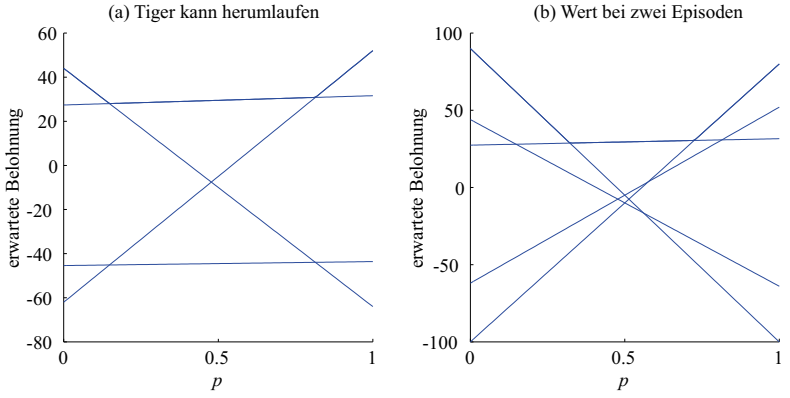


***Abb. 18.6*:** Die erwarteten Belohnungen ändern sich, wenn (a) der verborgene Zustand sich ändern kann und (b) wir Episoden der Länge zwei betrachten.

Dieses aktualisierte p muss dann in Gleichung 18.25 verwendet werden, wenn wir die beste Aktion wählen, nachdem wir zuvor a_S gewählt haben:

$$V' = \max \begin{pmatrix} -100p' & +80(1-p') \\ 33p' & +26(1-p') \\ 90p' & -100(1-p') \end{pmatrix} .$$

Abbildung 18.6b entspricht Abbildung 18.5d mit dem aktualisierten p'. Nun, da wir mit Episoden der Länge zwei planen, haben wir die beiden unmittelbaren Aktionen a_L und a_R und zusätzlich die Möglichkeiten zu warten und abzutasten. Wenn p sich ändert, führen wir die Aktion aus und erhalten die diskontierte Belohnung (Abbildung 18.6b):

$$V_2 = \max \begin{pmatrix} -100p & +80(1-p) \\ 90p & -100(1-p) \\ & \max V' - 1 \end{pmatrix} .$$

Wir sehen, dass Abbildung 18.6b besser ist als 18.5a; wenn falsche Aktionen zu hohen Bestrafungen führen können, ist es besser, das Urteil zu verschieben, nach zusätzlichen Informationen zu suchen und vorauszuplanen. Wir können längere Episoden betrachten, indem wir das iterative Aktualisieren von p fortsetzen, durch Subtrahieren von 1 diskontieren und die beiden unmittelbaren Aktionen einbeziehen, um $V_t, t > 2$ zu berechnen.

Der soeben diskutierte Algorithmus, bei dem der Wert durch stückweise lineare Funktionen repräsentiert wird, funktioniert nur, wenn die Anzahl der Zustände, Aktionen, Beobachtungen und Episodenlängen alle endlich sind. Selbst bei Anwendungen, für die eine dieser Größen nicht klein ist

oder kontinuierliche Werte annimmt, wird die Komplexität sehr groß und wir müssen auf Näherungsverfahren mit vertretbarer Komplexität zurückgreifen. Ein Überblick über solche Algorithmen wird in Hauskrecht 2000 sowie in Thrun, Burgard und Fox 2005 gegeben.

18.8 Anmerkungen

Weitere Informationen zum bestärkenden Lernen finden sich im Lehrbuch von Sutton und Barto (1998), in welchem diverse Aspekte, Lernalgorithmen und viele Applikationen Beachtung finden. Ein umfangreiches Tutorium bieten Kaelbling, Littman und Moore (1996). Über neuere Arbeiten zum bestärkenden Lernen in der Robotik einschließlich beeindruckender Anwendungen wird in Thrun, Burgard und Fox 2005 berichtet.

Dynamische Programmiermethoden werden bei Bertsekas (1987) und in Bertsekas und Tsitsiklis (1996) besprochen. Jaakkola, Jordan und Singh (1994) betrachten das TD(λ)- sowie das Q-Lernen als stochastische Approximationen an die dynamische Programmierung. Das bestärkende Lernen hat gegenüber dem klassischen dynamischen Programmieren zwei Vorteile: erstens kann sich während des Lernens auf die Teile des Raumes konzentriert werden, die wichtig sind, wohingegen alle anderen Teile ignoriert werden; zweitens können Methoden der Approximation von Funktionen eingesetzt werden, um Wissen zu repräsentieren, welches die Generalisierung und schnelleres Lernen gestattet.

Lernautomaten

Ein verwandtes Feld ist das der *Lernautomaten* (Narendra und Thathachar 1974), bei denen es sich um finite Zustandsmaschinen handelt, die durch Versuch und Irrtum lernen, um Probleme wie das des K-armigen Banditen zu lösen. Die hier vorliegende Ausgangslage ist auch Thema der optimalen Steuerung, wobei es einen Controller (Agenten) gibt, der Aktionen in einer Fabrik (Umwelt) ausführt, wodurch die Kosten minimiert werden (Maximierung der Belohnung).

Die früheste Verwendung der Methode der temporalen Differenz fand sich in Samuels Dameprogramm aus dem Jahre 1959 (Sutton und Barto 1998). Für jeweils zwei aufeinanderfolgende Stellungen in einem Spiel werden die zwei Brettzustände durch die Brettevaluierungsfunktion ausgewertet, wodurch dann eine Aktualisierung veranlasst wird, um die Differenz zu verringern. Es wurden viele Arbeiten zu (Brett-)Spielen vorangetrieben, da Spiele sowohl einfach definiert als auch herausfordernd sind. Ein Spiel wie Schach ist leicht zu simulieren: die erlaubten Züge sind formal und das Ziel klar definiert. Trotz der Einfachheit, das Spiel zu definieren, ist Spielen auf höchstem Niveau jedoch äußerst schwierig.

TD-Gammon-Programm

Eine der beeindruckendsten Applikationen des bestärkenden Lernens ist das *TD-Gammon-Programm*, welches lernt, Backgammon zu spielen, indem es gegen sich selbst spielt (Tesauro 1995). Dieses Programm ist dem vorherigen, ebenfalls durch Tesauro entwickelten Neurogammon-

Programm überlegen, welches auf überwachte Weise basierend auf Spielen von Experten trainiert wurde. Backgammon ist eine komplexe Aufgabe mit annähernd 10^{20} Zuständen, und aufgrund des Würfelns wird Zufälligkeit geschaffen. Unter Verwendung des TD(λ)-Algorithmus erreicht das Programm die höchste Spielstufe, nachdem es 1.500.000 Partien gegen eine Kopie von sich selbst gespielt hat.

Eine weitere interessante Anwendung ist das *Job Shop Scheduling*, also das Bestimmen eines Zeitplans für Aufgaben, welches zeitliche und ressourcenbedingte Einschränkungen befolgt (Zhang und Dietterich 1996). Einige Aufgaben müssen beendet werden, bevor andere beginnen und zwei Aufgaben mit den gleichen Ressourcenanforderungen können nicht simultan durchgeführt werden. Zhang und Dietterich nutzen das bestärkende Lernen, um schnell Zeitpläne zu finden, die den Einschränkungen entsprechen und kurz sind. Jeder Zustand ist ein Zeitplan, Aktionen sind Zeitplanmodifikationen und das Programm findet nicht nur einen guten Zeitplan, sondern einen Zeitplan für eine Klasse von verwandten Planungsproblemen.

In jüngster Vergangenheit wurden auch hierarchische Methoden entwickelt, bei denen das Problem in eine Menge an Unterproblemen zerlegt wird. Das hat den Vorteil, dass für das Unterproblem erlernte Taktiken für multiple Probleme genutzt werden können, wodurch das Erlernen des neuen Problems beschleunigt wird (Dietterich 2000). Jedes Unterproblem ist einfacher und das Lernen selbiger ist schneller. Der Nachteil besteht darin, dass sich bei ihrer Kombination die Taktik als suboptimal erweisen könnte.

Zwar sind Algorithmen des bestärkenden Lernens langsamer als Algorithmen des überwachten Lernens, jedoch ist offensichtlich, dass sie eine breitere Anwendungsvielfalt besitzen und das Potenzial vorweisen, bessere Lernmaschinen zu konstruieren (Ballard 1997). Sie bedürfen keinerlei Überwachung, was von Vorteil sein kann, denn sie werden dadurch nicht durch den Lehrer verzerrt. Zum Beispiel lieferte Tesauros TD-Gammon-Programm unter gewissen Umständen Züge, die sich denen der besten Spieler überlegen erwiesen. Das Gebiet des bestärkenden Lernens entwickelt sich rasch und es ist zu erwarten, dass wir in naher Zukunft weitere beeindruckende Ergebnisse zu sehen bekommen.

18.9 Übungen

1. Für die in Abbildung 18.7 gegebene Rasterwelt sei die Belohnung für das Erreichen des Ziels gleich 100 und $\gamma = 0{,}9$; berechnen Sie manuell $Q^*(s, a)$, $V^*(S)$ und die Aktionen der optimalen Taktik.

2. Nutzen Sie Q-Lernen für dieselbe Konfiguration aus Übung 1, um die optimale Taktik zu erlernen.

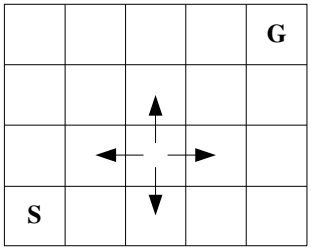


***Abb. 18.7*:** Die Rasterwelt. Der Agent kann sich beginnend bei S in eine der vier Himmelsrichtungen bewegen. Der Zielzustand ist G.

3. Wie verändert sich die optimale Taktik in Übung 1, wenn ein weiterer Zielzustand in der unteren rechten Ecke eingefügt wird? Was passiert, wenn ein Zustand der Belohnung -100 (ein sehr ungünstiger Zustand) in der unteren rechten Ecke definiert wird?

4. Statt $\gamma < 1$ zu setzen, können wir auch mit $\gamma = 1$ arbeiten, aber mit einer negativen Belohnung von $-c$ für alle Zwischenzustände (keine Zielzustände). Worin besteht der Unterschied?

5. Nehmen Sie für Übung 1 an, dass die Belohnung bei der Ankunft im Zielzustand normalverteilt ist mit Mittelwert 100 und Varianz 40. Nehmen Sie weiterhin an, dass die Aktionen dahingehend stochastisch sind, dass der Roboter sich beim Voranschreiten in eine Richtung mit Wahrscheinlichkeit 0,5 in die beabsichtigte Richtung bewegt und, dass er sich mit der Wahrscheinlichkeit von 0,25 in eine der lateralen Richtungen bewegt. Erlernen Sie für diesen Fall $Q(s, a)$.

6. Nehmen Sie an, dass wir die Wertfunktion für die Zustände $V(s)$ schätzen und, dass wir den TD(λ)-Algorithmus verwenden wollen. Leiten Sie die tabulare Wertiterations-Aktualisierung ab.

   Lösung: Der temporale Fehler zur Zeit t ist

   $$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \, .$$

   Alle Zustandswerte werden aktualisiert gemäß

   $$V(s) \leftarrow V(s) + \eta \delta_t e_t(s), \ \forall s \, ,$$

   wobei die Eignung der Zustände mit der Zeit abfällt:

   $$e_t(s) = \begin{cases} 1 & \text{falls } s = s_t, \\ \gamma\lambda e_{t-1}(s) & \text{sonst.} \end{cases}$$

7. Leiten Sie unter Verwendung von Gleichung 18.22 die Gleichungen zur Gewichtsaktualisierung ab, wenn ein mehrlagiges Perzeptron zur Schätzung von Q eingesetzt wird.

LÖSUNG: Nehmen wir der Einfachheit halber an, dass wir einen eindimensionalen Zustandsvektor s_t und einen eindimensionalen Aktionswert a_t haben, und setzen wir ein lineares Modell voraus:

$$Q(s,a) = w_1 s + w_2 a + w_3 \,.$$

Wir können die drei Parameter w_1, w_2, w_3 mittels Gradientenabstieg (Gleichung 16.21) aktualisieren:

$$\begin{aligned}
\Delta w_1 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\, s_t \,, \\
\Delta w_2 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\, a_t \,, \\
\Delta w_3 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \,.
\end{aligned}$$

Im Falle eines mehrlagigen Perzeptrons unterscheidet sich nur der letzte Term bei der Aktualisierung der Gewichte in allen Lagen.

Im Falle von Sarsa(λ) ist $\boldsymbol{e}$ dreidimensional: e^1 für w_1, e^2 für w_2 und e^3 für w_0. Wir aktualisieren die Eignungen (Gleichung 18.23):

$$\begin{aligned}
e_t^1 &= \gamma\lambda e_{t-1}^1 + s_t \,, \\
e_t^2 &= \gamma\lambda e_{t-2}^2 + a_t \,, \\
e_t^3 &= \gamma\lambda e_{t-3}^3 \,.
\end{aligned}$$

Die Gewichte aktualisieren wir unter Verwendung der Eignungen (Gleichung 18.22):

$$\begin{aligned}
\Delta w_1 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\, e_t^1 \,, \\
\Delta w_2 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\, e_t^2 \,, \\
\Delta w_3 &= \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\, e_t^3 \,.
\end{aligned}$$

8. Geben Sie ein Beispiel für eine Anwendung des bestärkenden Lernens, die durch einen POMDP modelliert werden kann. Definieren Sie die Zustände, Aktionen, Beobachtungen und die Belohnung.

9. Zeigen Sie, dass beim Tigerexperiment der Bereich, den wir noch einmal abtasten müssen, kleiner wird, wenn wir einen zuverlässigeren Sensor bekommen.

10. Bearbeiten Sie das Beispiel des Tigerexperiments noch einmal unter Verwendung der folgenden Belohnungsmatrix:

$r(A,Z)$	Tiger links	Tiger rechts
links öffnen	-100	$+10$
rechts öffnen	20	-100

18.10 Literaturangaben

Ballard, D. H. 1997. *An Introduction to Natural Computation.* Cambridge, MA: The MIT Press.

Bellman, R. E. 1957. *Dynamic Programming.* Princeton: Princeton University Press.

Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* New York: Prentice Hall.

Bertsekas, D. P., and J. N. Tsitsiklis. 1996. *Neuro-Dynamic Programming.* Belmont, MA: Athena Scientific.

Dietterich, T. G. 2000. „Hierarchical Reinforcement Learning with the MAXQ Value Decomposition." *Journal of Artificial Intelligence Research* 13: 227–303.

Hauskrecht, M. 2000. „Value-Function Approximations for Partially Observable Markov Decision Processes." *Journal of Artificial Intelligence Research* 13: 33–94.

Jaakkola, T., M. I. Jordan, and S. P. Singh. 1994. „On the Convergence of Stochastic Iterative Dynamic Programming Algorithms." *Neural Computation* 6: 1185–1201.

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra. 1998. „Planning and Acting in Partially Observable Stochastic Domains." *Artificial Intelligence* 101: 99–134.

Kaelbling, L. P., M. L. Littman, and A. W. Moore. 1996. „Reinforcement Learning: A Survey." *Journal of Artificial Intelligence Research* 4: 237–285.

Narendra, K. S., and M. A. L. Thathachar. 1974. „Learning Automata – A Survey." *IEEE Transactions on Systems, Man, and Cybernetics* 4: 323–334.

Sutton, R. S. 1988. „Learning to Predict by the Method of Temporal Differences." *Machine Learning* 3: 9–44.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement Learning: An Introduction.* Cambridge, MA: The MIT Press.

Tesauro, G. 1995. „Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38(3): 58–68.

Thrun, S., W. Burgard, and D. Fox. 2005. *Probabilistic Robotics.* Cambridge, MA: MIT Press.

Watkins, C. J. C. H., and P. Dayan. 1992. „Q-learning." *Machine Learning* 8: 279–292.

Zhang, W., and T. G. Dietterich. 1996. „High-Performance Job-Shop Scheduling with a Time-Delay TD(λ) Network.“ In *Advances in Neural Information Processing Systems 8,* ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 1024–1030. Cambridge, MA: The MIT Press.

19 Design und Analyse von Experimenten mit maschinellem Lernen

Wir diskutieren das Design von Experimenten auf der Basis maschinellen Lernens, um die Leistungsfähigkeit von Lernalgorithmen in der Praxis sowie die statistischen Test, welche die Ergebnisse dieser Experimente analysieren, bewerten und vergleichen zu können.

19.1 Einführung

In den vorangegangenen Kapiteln haben wir diverse Klassifikationsalgorithmen diskutiert und dabei festgestellt, dass bei einer gewissen vorliegenden Applikation mehr als einer anwendbar sein kann. Nun widmen wir uns zwei Fragen:

1. Wie können wir den erwarteten Fehler eines Klassifikationsalgorithmus an einem Problem bewerten? Das heißt, wenn wir einen Klassifikationsalgorithmus verwendet haben, um einen Klassifikator auf einer Datenmenge zu trainieren, die aus einer bestimmten Anwendung stammt, möchten wir mit ausreichender Sicherheit sagen können, dass die erwartete Fehlerrate bei zukünftiger Verwendung im realen Leben kleiner als beispielsweise 2% sein wird. Es ist zu klären, ob und wie dies möglich ist.

2. Wie können wir bei zwei vorliegenden Klassifikationsalgorithmen für eine Anwendung feststellen, ob einer einen geringeren Fehler verursacht als der andere? Die verglichenen Klassifikationsalgorithmen können unterschiedlich sein, zum Beispiel parametrisch versus nichtparametrisch, oder sie können unterschiedliche Hyperparametereinstellungen verwenden. Beispielsweise wollen wir für ein vorliegendes mehrlagiges Perzeptron (Kapitel 11) mit einmal vier verborgenen Einheiten und ein andermal mit acht verborgenen Einheiten in der Lage sein, zu sagen, welches der beiden den geringeren zu erwartenden Fehler aufweist. Oder beim k-Nächste-Nachbarn-Klassifikator (Kapitel 8) interessiert uns der beste Wert für k.

Wir können die Fehler nicht am Trainingsdatensatz betrachten und auf deren Basis eine Entscheidung treffen. Laut Definition ist die Fehlerrate am Trainingsdatensatz immer kleiner als die Fehlerrate an einem Testdatensatz, der Instanzen enthält, die während des Trainings nicht zur Verfügung standen. Auf ähnliche Weise können Trainingsfehler nicht für den Zweck des Vergleichs zweier Algorithmen herangezogen werden. Das liegt daran, dass das komplexere Modell mit seiner größeren Anzahl an Parametern über den Trainingsdatensatz fast immer weniger Fehler erzeugen wird als das einfache Modell.

Wie wir also schon mehrfach besprochen haben, benötigen wir einen Validierungsdatensatz, der sich vom Trainingsdatensatz unterscheidet. Allerdings kann selbst über eine Validierungsmenge ein einziger Durchlauf unter Umständen nicht ausreichend sein. Dafür gibt es zwei Gründe: Erstens können Trainings- und Validierungsdatensatz klein sein und Ausnahmeinstanzen enthalten, wie Rauschen und Ausreißer, welche uns in die Irre führen könnten. Zweitens kann die Lernmethode unter Umständen von anderen zufälligen Faktoren abhängen, wodurch die Generalisierung beeinflusst wird. Beispielsweise beeinflussen die Ausgangsgewichtungen bei einem mittels Rückpropagierung trainierten mehrlagigen Perzeptron aufgrund der Konvergenz des Gradientenabstiegs zum nächstliegenden lokalen Minimum die finalen Gewichte; und wenn bei exakt identischer Architektur und dem gleichen Trainingsdatensatz mit anderen Ausgangsgewichtungen begonnen wird, kann das Ergebnis aus multiplen möglichen finalen Klassifikatoren bestehen, die unterschiedliche Fehlerraten am selben Validierungsdatensatz haben. Somit wäre es wünschenswert, mehrere Durchläufe zu nutzen, um über solche Ursachen von Zufälligkeit hinweg Durchschnittswerte bilden zu können. Wenn wir einmal trainieren und validieren, dann können wir den Effekt solcher Faktoren nicht überprüfen; das ist nur dann zulässig, wenn die Lernmethode so kostspielig ist, dass sie nur einmal trainiert und validiert werden kann.

Wir nutzen einen *Klassifikationsalgorithmus* an einem Datensatz und generieren einen *Klassifikator*. Führen wir das Training einmal durch, so haben wir einen Klassifikator und einen Validierungsfehler. Um über die Zufälligkeit (bei Trainingsdaten, Ausgangsgewichtungen, etc.) hinweg den Durchschnitt zu bilden, nutzen wir denselben Algorithmus und generieren multiple Klassifikatoren. Wir testen diese Klassifikatoren an multiplen Validierungsdatensätzen und zeichnen eine Stichprobe von Validierungsfehlern auf. (Natürlich sollten alle Trainings- und Validierungsdatensätze aus derselben Anwendung entnommen werden.) Wir basieren unsere Evaluierung des Klassifikationsalgorithmus auf der *Verteilung* dieser Validierungsfehler. Wir können diese Verteilung nutzen, um den *erwarteten Fehler* des Klassifikationsalgorithmus für das jeweilige Problem zu bewerten, oder um sie mit der Verteilung der Fehlerrate eines anderen Klassifikationsalgorithmus zu vergleichen.

ERWARTETER FEHLER

Bevor wir betrachten, wie dies durchgeführt wird, sollten einige Punkte unbedingt noch einmal betont werden:

1. Welche Schlussfolgerung auch immer wir aus unserer Analyse ziehen, so sollten wir bedenken, dass sie von dem Datensatz abhängt, der uns vorgelegt wird. Wir vergleichen die Klassifikationsalgorithmen nicht auf domänenunabhängige Weise, sondern im Rahmen einer bestimmten Applikation. Wir treffen keinerlei Aussage hinsichtlich der erwarteten Fehlerrate eines Lernalgorithmus, noch vergleichen wir generell einen Lernalgorithmus mit einem anderen Algorithmus. Jedes Ergebnis, zu dem wir kommen, ist immer nur für die jeweilige Anwendung zutreffend und nur insofern, wie jene Anwendung in der uns vorliegenden Stichprobe repräsentiert ist. Und tatsächlich besagt das *Nichts-ist-umsonst-Theorem* (auch *No-Free-Lunch-Theorem,* Wolpert 1995), dass es keinen „besten" Lernalgorithmus gibt. Für jeden Lernalgorithmus existiert ein Datensatz, bei dem er sehr genau arbeitet und ein anderer Datensatz, bei dem er eher miserable Ergebnisse liefert. Wenn wir sagen, dass ein Klassifikationsalgorithmus gut ist, dann quantifizieren wir lediglich, wie gut seine induktive Verzerrung mit den Eigenschaften der Daten übereinstimmt.

   NICHTS-IST-UMSONST-THEOREM

2. Die Zerlegung eines gegebenen Datensatzes in eine Zahl an Paaren aus Trainings- und Validierungsdaten geschieht nur zu Testzwecken. Sobald alle Tests beendet sind und wir unsere Entscheidung hinsichtlich der endgültigen Methode oder Hyperparameter getroffen haben, können wir zum Training des finalen Klassifikators alle zugeordneten Daten nutzen, die wir vorher zum Training oder zur Validierung verwendet haben.

3. Weil wir auch den Validierungsdatensatz (bzw. die Validierungsdatensätze) zu Testzwecken verwenden, zum Beispiel um den besseren von zwei Klassifikationsalgorithmen zu wählen oder um zu entscheiden, wann mit dem Lernen aufzuhören ist, wird er effektiv zu einem Teil der von uns genutzten Daten. Wenn wir uns nach all solchen Tests bezüglich eines bestimmten Klassifikationsalgorithmus entscheiden und seine erwartete Fehlerrate bestimmen wollen, dann sollten wir einen separaten *Testdatensatz* zu diesem Zweck benutzen, der während des Trainings dieses finalen Systems nicht in Gebrauch war. Diese Daten sollten nie vorher zum Training oder zur Validierung genutzt worden sein und außerdem einen Umfang haben, der eine sinnvolle Fehlerschätzung ermöglicht. Bei einem gegebenen Datensatz sollten wir also zuerst einen Teil davon als Testdatensatz beiseite legen und den Rest zum Training und zur Validierung verwenden. Typischerweise können wir ein Drittel der Stichprobe als Testdatensatz zurücklegen, und dann die zwei verbliebenen Drittel für die Kreuzvalidierung nutzen, um multiple Paare von Trainings-

und Validierungsdaten zu generieren, wie wir im folgenden Abschnitt sehen werden. Somit wird der Trainingsdatensatz verwendet, um die Parameter für einen gegebenen bestimmten Lernalgorithmus und eine Modellstruktur zu optimieren; der Validierungsdatensatz wird genutzt, um die Hyperparameter des Lernalgorithmus oder der Modellstruktur zu optimieren; der Testdatensatz kommt ganz am Ende ins Spiel, sobald diese beiden optimiert worden sind. Mit einem MLP zum Beispiel wird der Trainingsdatensatz genutzt, um die Gewichte zu optimieren, und der Validierungsdatensatz dient der Entscheidung hinsichtlich der Anzahl an verborgenen Einheiten, der Länge des Trainings, der Lernrate, und so weiter. Ist die beste MLP-Konfiguration einmal gewählt, wird die zugehörige finale Fehlerrate am Testdatensatz berechnet. Beim k-NN wird die Trainingsmenge als Lookup-Tabelle gespeichert; wir optimieren das Distanzmaß und k am Validierungsdatensatz und testen schließlich am Testdatensatz.

4. In diesem Kapitel vergleichen wir Klassifikationsalgorithmen anhand ihrer Fehlerraten, jedoch sollte bedacht werden, dass im realen Leben der Fehler nur eins von mehreren Kriterien ist, die unsere Entscheidung beeinflussen. Einige andere Kriterien umfassen (Turney 2000):

   - Risiken, wenn Fehler mit Hilfe von Verlustfunktionen, statt mit 0/1-Verlust generalisiert werden (Abschnitt 3.3),
   - Komplexität der Rechenzeit und des Speicherbedarfs des Trainings,
   - Komplexität der Rechenzeit und des Speicherbedarfs des Testens,
   - Interpretierbarkeit, sprich, ob die Methode die Extraktion von Wissen ermöglicht, welches durch Experten überprüft und validiert werden kann, und
   - die einfache Programmierbarkeit.

   Die relative Wichtigkeit dieser Faktoren ändert sich in Abhängigkeit von der Applikation. Soll das Training zum Beispiel nur einmal werkseitig durchgeführt werden, dann ist die Komplexität der Rechenzeit und des Speicherbedarfs des Trainings unwichtig; wird Adaptivität während der Verwendung gefordert, dann wird sie jedoch wichtig. Die meisten der Lernalgorithmen nutzen 0/1-Fehler und sehen den Fehler als das einzige zu minimierende Kriterium; in neuester Zeit wurden auch *kostensensitive* Lernvarianten dieser Algorithmen entworfen, um andere Kostenkriterien in die Erwägungen einzubeziehen.

KOSTENSENSITIVES LERNEN

Wenn wir einen Lerner auf einem Datensatz trainieren, seine Genauigkeit auf einem Validierungsdatensatz testen und dann Schlüsse ziehen, dann

können wir das, was wir tun, Experimentieren nennen. Die Statistik definiert eine Methodik für das korrekte Design von Experimenten und die Analyse der gesammelten Daten, die uns in die Lage versetzt, signifikante Schlussfolgerungen ableiten zu können (Montgomery 2005). In diesem Kapitel beschäftigen wir uns damit, wie diese Methodik im Zusammenhang mit dem maschinellen Lernen angewendet werden kann.

19.2 Faktoren, Antwort und Strategie beim Experimentieren

Wie in anderen Zweigen der Wissenschaft und des Ingenieurwesens führen wir auch beim maschinellen Lernen Experimente durch, um Informationen über den zu untersuchenden Gegenstand zu gewinnen. In unserem Fall ist dieser Gegenstand ein Lerner, der, nachdem er auf einer Datenmenge trainiert wurde, für eine gegebene Eingabe eine Ausgabe generiert. Ein *Experiment* ist ein Test oder eine Serie von Tests, bei denen wir mit den *Faktoren* spielen, welche die Ausgabe beeinflussen. Diese Faktoren können der verwendete Algorithmus, die Trainingsmenge, Eingabemerkmale usw. sein, und wir beobachten, wie sich die *Antwort* ändert, um daraus Informationen zu extrahieren. Das Ziel kann etwa darin bestehen, die wichtigsten Faktoren zu identifizieren, unwichtige herauszufiltern oder diejenige Konfiguration der Faktoren zu finden, welche die Antwort – beispielsweise die Genauigkeit der Klassifikation auf einer gegebenen Testmenge – optimiert.

Experiment

Unser Ziel ist es, Experimente auf der Basis maschinellen Lernens zu planen und durchzuführen sowie die aus den Experimenten resultierenden Daten zu analysieren. Dies soll uns in die Lage versetzen, den Einfluss des Zufalls zu eliminieren und Schlussfolgerungen zu erhalten, die als *statistisch signifikant* angesehen werden können. Beim maschinellen Lernen streben wir einen Lerner an, der die größtmögliche Genauigkeit mit minimaler Komplexität verbindet (so dass die Implementierung kostengünstig ist) und zudem robust ist, d. h., nur minimal von externen Schwankungen beeinflusst wird.

Ein trainierter Lerner kann wie in Abbildung 19.1 dargestellt werden. Er liefert uns für eine Eingabe eine Ausgabe, beispielsweise einen Klassencode, und diese hängt von zwei Arten von Faktoren ab. Die *kontrollierbaren Faktoren* sind, wie der Name schon sagt, diejenigen, die wir steuern können. Der grundlegendste dieser Faktoren ist der verwendete Lernalgorithmus. Außerdem gibt es die Hyperparameter des Algorithmus, so zum Beispiel die Anzahl der verborgenen Einheiten im Falle eines mehrlagigen Perzeptrons, k für den k-NN-Algorithmus, C für Support-Vektor-Maschinen usw. Die verwendete Datenmenge und die Repräsentation der Eingabe, d. h. die Art und Weise ihrer Codierung, sind weitere kontrollierbare Faktoren.

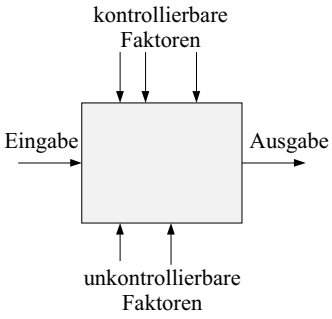


Abb. 19.1: Der Prozess generiert für eine gegebene Eingabe eine Ausgabe, wobei er von kontrollierbaren und unkontrollierbaren Faktoren beeinflusst wird.

Neben diesen gibt es auch *unkontrollierbare Faktoren,* über die wir keine Kontrolle haben und die dem Prozess eine unerwünschte Variabilität verleihen können. Wir wollen natürlich vermeiden, dass solche Faktoren unsere Entscheidungen beeinflussen. Zu den unkontrollierbaren Faktoren gehören das in den Daten vorhandene Rauschen, die spezielle Trainingsmenge, falls wir sie als Stichprobe aus einer großen Datenmenge gezogen haben oder Zufälligkeiten beim Optimierungsprozess (etwa die Wahl des Anfangszustands beim Gradientenabstieg für mehrlagige Perzeptronen).

Wir verwenden die Ausgabe, um die *Antwortvariable* zu erzeugen, beispielsweise den mittleren Fehler auf einer Testmenge oder das erwartete Risiko bei der Verwendung einer Verlustfunktion oder irgendein anderes Maß wie die Präzision oder die Trefferquote. Damit werden wir uns in Kürze genauer beschäftigen.

Wenn mehrere Faktoren gegeben sind, müssen wir die beste Konfiguration für die bestmögliche Antwort finden oder im allgemeinen Fall ihren Einfluss auf die Antwortvariable bestimmen. Wir können zum Beispiel die Hauptkomponentenanalyse (PCA) verwenden, um die Dimensionalität auf d zu reduzieren, bevor wir einen k-NN-Klassifikator anwenden. Die beiden Faktoren sind d und k, und die Aufgabe besteht darin zu entscheiden, welche Kombination von d und k zur besten Leistungsfähigkeit führt. Oder wir verwenden einen SVM-Klassifikator mit Gauß-Kernel und wollen dann für den Regularisierungsparameter C und die Streuung der Gauß-Variablen, s^2, die optimale Feinabstimmung finden.

Strategien des Experimentierens

Es gibt verschiedene *Strategien des Experimentierens,* die in Abbildung 19.2 schematisch dargestellt sind. Beim *besten Tipp* starten wir mit irgendeiner Konfiguration der Parameter, von der wir glauben, dass sie gut ist. Wir testen diese die Antwort und spielen solange an einem einzelnen Faktor (oder einigen wenigen gleichzeitig) herum und testen jede Kombination, bis wir zu einem Zustand kommen, den wir als ausreichend gut betrachten. Wenn der Experimentator in Bezug auf den Prozess eine gute Intuition hat, kann das gut funktionieren; allerdings ist zu beachten, dass es keine systematische Vorgehensweise gibt, um die Faktoren zu modifizieren, und wenn wir aufhören, gibt es keine Garantie dafür, dass wir die beste Konfiguration gefunden haben.

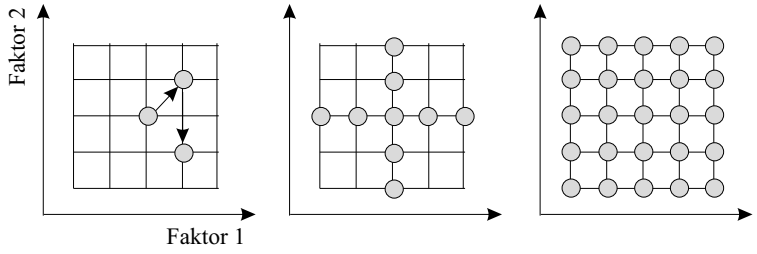


Abb. 19.2: Unterschiedliche Strategien des Experimentierens mit zwei Faktoren und jeweils fünf Einstellmöglichkeiten.

Eine andere Strategie besteht darin, *immer nur einen Faktor* zu modifizieren, während für die übrigen Faktoren Defaultwerte angenommen werden. Diese Vorgehensweise hat allerdings einen großen Nachteil, und zwar die Annahme, dass es keine *Wechselwirkungen* zwischen den Faktoren gibt. Das ist selbstverständlich nicht immer der Fall. Bei der PCA/k-NN-Kaskade, die wir weiter vorn diskutiert haben, definiert jede Wahl von d einen anderen Eingaberaum für k-NN, wobei jeweils ein anderer k-Wert passend sein kann.

FAKTORIELLES DESIGN

Der richtige Ansatz ist ein *faktorielles Design,* was bedeutet, dass die Faktoren gemeinsam variiert werden, anstatt immer nur einen zu variieren und die anderen festzuhalten. Dies wird auch *Gittersuche* genannt. Mit F Faktoren und jeweils L Einstellmöglichkeiten benötigt man für die Suche bei separater Variation der Faktoren die Zeit $\mathcal{O}(L \cdot F)$, während ein faktorielles Experiment die Zeit $\mathcal{O}(L^F)$ braucht.

19.3 Antwortflächenmethode

Eine Möglichkeit, die Anzahl der erforderlichen Durchläufe zu senken, ist ein fraktionales faktorielles Design, bei dem wir nur mit einer Teilmenge arbeiten. Eine andere Möglichkeit besteht darin, Wissen auszunutzen, das in früheren Durchläufen zusammengetragen wurde, um Konfigurationen zu schätzen, die mit hoher Wahrscheinlichkeit eine gute Antwort produzieren. Wenn wir bei der Suche mit separater Variation der Faktoren annehmen können, dass die Antwort typischerweise quadratisch ist (mit einem eindeutigen Maximum, wenn wir annehmen, dass wir einen Antwortwert wie die Genauigkeit maximieren), dann können wir, anstatt alle Werte durchzuprobieren, eine iterative Prozedur konstruieren. Dabei starten wir mit einer geeigneten Anfangsfunktion, passen eine quadratische Funktion an, bestimmen analytisch ihr Maximum, nehmen dieses als die nächste Schätzung, führen damit ein Experiment aus, fügen die daraus resultierenden Daten zur Stichprobe hinzu und fahren dann

solange mit dem Anpassen und der Stichprobenerhebung fort, bis wir keine weitere Verbesserung mehr erreichen.

Antwortflächenmethode

Wenn es viele Faktoren gibt, wird dieser Ansatz zur *Antwortflächenmethode* verallgemeinert. Dabei wird versucht, eine parametrische Antwortfunktion an die Faktoren anzupassen:

$$r = g(f_1, f_2, \ldots, f_F|\phi)$$

Hierin ist r die Antwort und die $f_i, i = 1, \ldots, F$ sind die Faktoren. Diese an die gegebenen Parameter ϕ angepasste parametrische Funktion ist unser empirisches Modell zum Schätzen der Antwort für eine spezielle Konfiguration der (kontrollierbaren) Faktoren. Der Effekt der unkontrollierbaren Faktoren wird durch ein Rauschen modelliert. $g(\cdot)$ ist ein (typischerweise quadratisches) Regressionsmodell und nach einer kleinen Zahl von Durchläufen um eine Defaultlinie (definiert durch eine sogenannte *Designmatrix*) werden wir genug Daten haben, um $g(\cdot)$ anzupassen. Anschließend können wir analytisch die Werte der f_i berechnen, für die g maximal wird. Diese nehmen wir als unsere nächste Schätzung, führen damit das Experiment durch, erhalten eine Dateninstanz, fügen sie zur Stichprobe hinzu, passen g erneut an usw., bis Konvergenz erreicht ist. Ob dieser Ansatz gut funktioniert oder nicht, hängt davon ab, ob sich die Antwort tatsächlich als quadratische Funktion (mit eindeutigem Maximum) der Faktoren schreiben lässt.

19.4 Randomisieren, Wiederholen und Blocken

Kommen wir nun zu den drei grundlegenden Prinzipien des Designs von Experimenten:

Randomisieren

- *Randomisieren* bedeutet, dass die Reihenfolge der Durchläufe zufällig gewählt wird, so dass die Ergebnisse unabhängig sind. Dies ist ein typisches Problem in realen Experimenten, also solchen, die mit physikalischen Objekten durchgeführt werden. Maschinen zum Beispiel benötigen eine gewisse Zeit um warmzulaufen, bevor sie ihren normalen Betriebsmodus erreichen, und daher sollten Tests in einer zufälligen Reihenfolge durchgeführt werden, um die Ergebnisse nicht zu verzerren. Bei Software-Experimenten stellt die Reihenfolge im Allgemeinen kein Problem dar.

Wiederholen

- *Wiederholen* bedeutet, dass das Experiment für die gleiche Konfiguration der (kontrollierbaren) Faktoren mehrfach durchgeführt werden sollte, damit sich der Einfluss der unkontrollierbaren Faktoren herausmittelt. Beim maschinellen Lernen wird das üblicherweise

dadurch erreicht, dass man denselben Algorithmus auf einer Reihe von neu gemischten Versionen der gleichen Datenmenge laufen lässt. Dieses als Kreuzvalidierung bezeichnete Prinzip werden wir in Abschnitt 19.6 diskutieren. Indem wir beobachten, wie die Antwort mit den verschiedenen Wiederholungen variiert, können wir eine Schätzung für den experimentellen Fehler erhalten (also für den Effekt der unkontrollierbaren Faktoren). Mit diesen können wir dann bestimmen, wie groß Differenzen sein sollten, um als *statistisch signifikant* angesehen zu werden.

- *Blocken* wird angewendet, um die Schwankungen aufgrund von Störfaktoren zu reduzieren oder zu eliminieren, d. h. von solchen Faktoren, die die Antwort beeinflussen, uns aber nicht interessieren. Beispielsweise können Defekte, die bei der Fertigung eines Produktes entstehen, auch auf unterschiedliche Chargen des Rohmaterials zurückzuführen sein, und dieser Effekt sollte von den kontrollierbaren Faktoren des Fertigungsprozesses (Maschinen, Arbeitskräfte usw.) isoliert werden. Beim Experimentieren auf der Basis des maschinellen Lernens arbeiten wir mit Stichproben und verwenden unterschiedliche Teilmengen der Daten für die verschiedenen Wiederholungen. Daher müssen wir sicherstellen, dass zum Beispiel beim Vergleichen von Lernalgorithmen alle Algorithmen die gleichen Stichprobenmengen verwenden, denn sonst würden die Unterschiede in den Genauigkeiten nicht allein von den Algorithmen, sondern auch von den unterschiedlichen Stichprobenmengen abhängen. Um allein den Unterschied aufgrund der Algorithmen messen zu können, sollten die bei den einzelnen Durchläufen verwendeten Trainingsmengen identisch sein – das ist es, was wir mit Blocken meinen. Wenn man in der Statistik zwei Populationen betrachtet, nennt man dieses Prinzip auch *Paarung* und verwendet es bei *gepaarten Tests.* BLOCKEN PAARUNG

19.5 Richtlinien für Experimente mit maschinellem Lernen

Bevor wir mit dem Experimentieren beginnen, sollten wir eine gute Vorstellung von dem Gegenstand unserer Untersuchung haben. Ebenso sollten wir wissen, wie die Daten erhoben werden und wir sie analysieren wollen. Die Schritte sind beim maschinellen Lernen die gleichen wie bei jeder anderen Form des Experimentierens (Montgomery 2005). Dabei ist es nicht wichtig, ob es sich bei der Aufgabe um eine Klassifikation oder eine Regression handelt, ebenso wenig ob wir es mit nichtüberwachtem Lernen oder bestärkendem Lernen zu tun haben. Die hier vorgestellten Richtlinien sind allgemein anwendbar; der Unterschied liegt allein in der Stichprobenverteilung der Anwortdaten.

A. Zweck der Studie

Am Anfang muss immer die klare Formulierung des Problems stehen, und es muss definiert werden, was die Ziele sind. Beim maschinellen Lernen kann es verschiedene Möglichkeiten geben. Wie wir bereits weiter vorn erörtert hatten, ist eine dieser Möglichkeiten, dass wir an der Bewertung des erwarteten Fehlers (oder eines anderen Maßes für die Antwort) eines Lernalgorithmus für ein spezielles Problem interessiert sind. Wir prüfen dann zum Beispiel, ob der Fehler kleiner als eine akzeptable Schwelle ist.

Wenn zwei Lernalgorithmen und ein spezielles Problem auf einer Datenmenge gegeben sind, dann wollen wir herausfinden, welcher der beiden Algorithmen den kleineren Generalisierungsfehler hat. Die beiden Algorithmen können tatsächlich verschieden sein; es ist aber auch möglich zwei verschiedene Versionen des gleichen Algorithmus zu bewerten, wobei einer einen Verbesserungsvorschlag enthält, etwa indem er mit einem besseren Merkmalsextraktor arbeitet.

Im allgemeinen Fall können wir mehr als zwei Lernalgorithmen betrachten. Unser Ziel ist es dann entweder, den Algorithmus mit dem kleinsten Fehler auszuwählen oder die Algorithmen für eine gegebene Datenmenge anhand des Fehlers in eine Reihenfolge zu bringen.

In einem noch allgemeineren Szenario betrachten wir nicht mehr eine einzelne Datenmenge, sondern wollen zwei oder mehrere Algorithmen auf zwei oder mehr Datenmengen vergleichen.

B. Auswahl der Antwortvariable

Wir müssen entscheiden, was wir als Maß für die Qualität verwenden wollen. Meist wird hierfür der Fehler verwendet, worunter wir bei der Klassifikation den Fehlklassifikationsfehler und bei der Regression den mittleren quadratischen Fehler verstehen. Wir können auch irgendeine Variante benutzen, zum Beispiel indem wir die 0/1-Verlustfunktion auf eine beliebige Verlustfunktion verallgemeinern und so ein Maß für das Risiko erhalten. Beim Information Retrieval verwenden wir Maße wie die Präzision und die Trefferquote, die wir in Abschnitt 19.7 behandeln werden. In einer kostensensitiven Anwendungssituation wird nicht nur die Ausgabe betrachtet, sondern es werden auch Systemparameter wie die Komplexität berücksichtigt.

C. Wahl der Faktoren und Einstellmöglichkeiten

Was die Faktoren in einer Studie sind, hängt vom Zweck der Studie ab. Wenn wir einen bestimmten Algorithmus wählen und das Ziel verfolgen, die besten Hyperparameter zu finden, dann sind diese die Faktoren. Wenn wir Algorithmen vergleichen, ist der Lernalgorithmus ein Faktor. Wenn wir verschiedene Datenmengen haben, sind auch diese Faktoren.

Die Einstellmöglichkeiten eines Faktor sollten sorgfältig gewählt werden, um tatsächlich eine gute Konfiguration zu finden und unnötig langes Experimentieren zu vermeiden. Nach Möglichkeit sollte man immer versuchen, die Einstellmöglichkeiten der Faktoren zu normieren. Beispielsweise kann man beim Optimieren von k in einem k-NN-Algorithmus Werte wie 1, 3, 5 usw. ausprobieren. Dagegen sollten wir es beim Optimieren der Streuung h von Parzen-Fenstern nicht mit absoluten Werten wie 1,0, 2,0 usw. versuchen, da diese Größe von der Skala der Eingabe abhängt. Besser ist es in solch einem Fall, eine Statistik zu finden, die ein Indikator für die Skala ist – beispielsweise den mittleren Abstand zwischen einer Instanz und ihrem nächsten Nachbarn – und dann für h verschiedene Vielfache dieser Statistik auszuprobieren.

Obwohl die bei früheren Experimenten gesammelten Erfahrungen allgemein von Vorteil sind, ist es doch ebenso wichtig, alle Faktoren und deren Einstellmöglichkeiten zu untersuchen, die für das aktuelle Problem von Bedeutung sein könnten und sich nicht zu sehr von früheren Erfahrungen beeinflussen zu lassen.

D. Design des Experiments

Es ist immer besser, mit einem faktoriellen Design zu arbeiten, es sei denn wir können sicher sein, dass die Faktoren nicht miteinander interagieren – was sie aber meistens tun. Die Anzahl der Wiederholungen hängt von der Größe der Datenmenge ab; sie kann klein gehalten werden, wenn die Datenmenge groß ist. Wir werden diesen Zusammenhang im nächsten Abschnitt genauer erörtern, wenn wir uns mit dem Resampling beschäftigen. Allerdings generieren wenige Wiederholungen auch wenige Daten, und das wird das Vergleichen von Verteilungen schwierig machen. Insbesondere bei parametrischen Tests kann es sein, dass die Annahme der Normalverteilung nicht mehr haltbar ist.

Wenn wir eine Datenmenge gegeben haben, dann wählen wir im Allgemeinen einen gewissen Teil als Testmenge und verwenden den Rest für das Training und die Validierung, wobei wir wahrscheinlich oftmals ein Resampling durchführen. Dabei ist es von Bedeutung, wie diese Unterteilung vorgenommen wird. In der Praxis führt die Verwendung kleiner Datenmengen zu Antworten mit hoher Varianz, so dass wir keine signifikanten Unterschiede sehen können und die Ergebnisse keine Schlüsse zulassen.

Außerdem ist es wichtig, so gut es geht auf synthetische Daten zu verzichten und möglichst nur Datenmengen aus der realen Welt zu verwenden, die unter realistischen Bedingungen erhoben wurden. Didaktische ein- oder zweidimensionale Datenmengen können dazu beitragen, die Intuition zu schärfen, doch das Verhalten der Algorithmen kann in hochdimensionalen Räumen ganz anders aussehen.

E. Durchführung des Experiments

Bevor man ein großes faktorielles Experiment mit vielen Faktoren und Einstellmöglichkeiten startet, ist es sinnvoll, zunächst einige Versuchsdurchläufe mit zufällig gewählten Einstellungen durchzuführen, um zu testen, ob alles so läuft wie erwartet. Bei einem großen Experiment ist es außerdem immer eine gute Idee, Zwischenergebnisse zu speichern (oder die Startwerte des Zufallszahlengenerators), so dass ein Teil des Experiments bei Bedarf wiederholt werden kann. Alle Ergebnisse sollten reproduzierbar sein. Bei der Durchführung eines großen Experiments mit vielen Faktoren und Einstellmöglichkeiten sollte man sich der möglichen negativen Effekte der Software-Alterung bewusst sein.

Es ist wichtig, dass der Experimentator während des Experimentierens unvoreingenommen bleibt. Wenn man beispielsweise den Algorithmus, den man selbst favorisiert, mit einem anderen vergleicht, muss man beide selbstverständlich mit der gleichen Sorgfalt betrachten. Bei großangelegten Studien könnte man sogar ins Auge fassen, dass die Tester andere Personen sein müssen als die Entwickler.

Man sollte zudem der Versuchung widerstehen, seine eigene „Bibliothek“ zu schreiben, und stattdessen so weit wie möglich Code aus verfügbaren Quellen nutzen, denn bei diesem kann man davon ausgehen, dass er besser getestet und optimiert ist.

Wie überall in der Softwareentwicklung können die Vorteile einer guten Dokumentation gar nicht hoch genug eingeschätzt werden, besonders dann, wenn man im Team arbeitet. Alle Methoden, die für die Entwicklung qualitativ hochwertiger Software entwickelt wurden, sollten auch bei Experimenten auf der Basis des maschinellen Lernens eingesetzt werden.

F. Statistische Analyse der Daten

Statistische Analyse bedeutet, die Daten in einer Weise zu analysieren, die sicherstellt, dass die gezogenen Schlüsse weder subjektiv noch zufällig sind. Wir entwerfen die Fragen, die wir beantwortet haben wollen, im Rahmen des Testens von Hypothesen, und prüfen dann, ob die Stichprobe die Hypothese stützt. Aus der Frage „Ist A ein genauerer Algorithmus als B?“ wird so zum Beispiel die Hypothese „Können wir sagen, dass der mittlere Fehler von Lernern, die durch A trainiert wurden, signifikant kleiner ist als der mittlere Fehler von Lernern, die durch B trainiert wurden?“

Wie immer ist eine visuelle Analyse hilfreich. Außerdem können wir Histogramme der Fehlerverteilungen, Whisker-Box-Plots, Range-Plots und andere nützliche Diagramme verwenden.

G. Schlussfolgerungen und Empfehlungen

Nachdem alle Daten erhoben und analysiert sind, können wir objektive Schlussfolgerungen ziehen. Eine Schlussfolgerung, die oft von Interesse ist, betrifft die Frage, ob weitere Experimente nötig sind. Die meisten statistischen Studien, und damit auch solche, die maschinelles Lernen oder Data Mining nutzen, sind iterativ angelegt. Aus diesem Grund schöpfen wir am Anfang niemals alle Mäglichkeiten aus. Es wird empfohlen, im ersten Experiment nicht mehr als 25 Prozent der verfügbaren Ressourcen zu investieren (Montgomery 2005). Die ersten Durchläufe dienen nur der Erkundung. Auch deshalb sollte man nicht mit zu hohen Erwartungen – oder Versprechungen gegenüber dem Chef oder Mentor – starten.

Wir sollten stets im Kopf behalten, dass ein statistischer Test uns nicht sagt, ob unsere Hypothese richtig oder falsch ist, sondern nur, wie sehr die Stichprobe mit der Hypothese im Einklang steht. Es gibt immer ein Risiko, dass wir kein schlüssiges Ergebnis bekommen oder dass unsere Schlussfolgerungen falsch sind, besonders wenn die Datenmenge klein und verrauscht ist.

Wenn unsere Erwartungen nicht erfüllt werden, ist es in der Regel hilfreich zu klären, warum das so ist. Beispielsweise kann uns die Untersuchung, warum der von uns favorisierte Algorithmus A in einigen Fällen so furchtbar schlecht funktioniert hat, auf eine großartige Idee bringen, wie eine verbesserte Version von A aussehen könnte. Alle Verbesserungen sind das Ergebnis von Mängeln der Vorgängerversion, und daher sind entdeckte Mängel immer auch Hinweise, dass man es besser machen kann!

Wir sollten uns allerdings erst an den nächsten Schritt, das Testen der verbesserten Version, machen, wenn wir sicher sind, dass wir die aktuellen Daten vollständig analysiert haben und alles aus ihnen gelernt haben, was man daraus lernen kann. Ideen sind billig, und sie sind nutzlos, solange sie nicht getestet sind – das Testen aber ist teuer.

19.6 Kreuzvalidierung und Resampling-Methoden

Für Wiederholungszwecke benötigen wir in erster Linie eine Anzahl an Paaren von Trainings- und Validierungsdaten aus einer Datenmenge $\mathcal{X}$ (nachdem wir einen Teil für die Testmenge herausgenommen haben). Um sie zu erhalten, können wir die Stichprobe bei ausreichender Größe zufällig in K Teile zerlegen, dann jeden Teil zufällig in zwei Teile aufspalten und einen Teil für das Training und den anderen für die Validierung nutzen. K ist typischerweise 10 oder 30. Leider sind Datensätze niemals groß genug, um so vorgehen zu können. Somit müssen wir unser Bestes bei den kleinen Datensätzen geben. Dafür nutzen wir dieselben Daten wiederholt, nur mit unterschiedlichen Aufspaltungen; das bezeichnet man

KREUZVALIDIERUNG

als *Kreuzvalidierung*. Der Knackpunkt besteht aber darin, dass dadurch die Fehlerprozentsätze untereinander in Abhängigkeit gebracht werden, da diese verschiedenen Mengen Daten gemeinsam haben.

Für einen gegebenen Datensatz $\mathcal{X}$ wollen wir also aus diesem Satz K Paare von Trainings-/Validierungsdaten, $\{\mathcal{T}_i, \mathcal{V}_i\}_{i=1}^K$, generieren. Wir wollen, dass die Trainings- und Validierungsmengen so groß wie möglich sind, so dass die Fehlerschätzungen robust sind; gleichzeitig wollen wir aber die Überschneidungen zwischen unterschiedlichen Mengen so klein als möglich halten. Außerdem müssen wir sicherstellen, dass Klassen in den korrekten Proportionen repräsentiert werden, wenn Teilmengen von Daten zurückgehalten werden, um nicht die a-priori-Wahrscheinlichkeiten der Klassen zu verzerren; hierbei spricht man von *Stratifizierung*. Wenn eine Klasse 20% der Beispiele im gesamten Datensatz hat, dann sollte sie auch in allen aus dem Datensatz gezogenen Stichproben ungefähr 20% der Beispiele besitzen.

STRATIFIZIERUNG

19.6.1 K-fache Kreuzvalidierung

K-FACHE KREUZVALIDIERUNG

Bei der *K-fachen Kreuzvalidierung* wird der Datensatz $\mathcal{X}$ zufällig in K gleichgroße Teile, $\mathcal{X}_i, i = 1, \ldots, K$, zerlegt. Um jedes Paar zu generieren, behalten wir einen der K Teile als Validierungsdatensatz zurück und kombinieren die verbleibenden $K-1$ Teile, um den Trainingsdatensatz zu formen. Führen wir dies K Mal durch und lassen wir jedes Mal einen anderen der K Teile aus, so erhalten wir K Paare:

$$\begin{array}{ll} \mathcal{V}_1 = \mathcal{X}_1 & \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \mathcal{V}_2 = \mathcal{X}_2 & \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K \\ \quad \vdots & \\ \mathcal{V}_K = \mathcal{X}_K & \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1} \,. \end{array}$$

Hierbei tun sich zwei Probleme auf: erstens erlauben wir kleine Validierungsdatensätze, um die Trainingsmenge groß zu halten. Zweitens überlappen sich Trainingsdatensätze doch deutlich, sprich, beliebige zwei Trainingsdatensätze haben $K-2$ Teile gemeinsam.

K ist typischerweise 10 oder 30. Mit zunehmendem K erhöht sich der Prozentsatz an Trainingsinstanzen und wir erhalten robustere Schätzer der Modellparameter, jedoch verkleinert sich dadurch die Validierungsmenge. Weiterhin entstehen Kosten, um den Klassifikator K Mal zu trainieren, welche sich mit wachsendem K auch erhöhen. Mit steigendem N kann K kleiner sein; ist N klein, dann sollte K groß sein, um ausreichend große Trainingsdatensätze zu ermöglichen. Ein extremer Fall der K-fachen Kreuzvalidierung ist der *Leave-One-Out-Fall*, bei dem bei einem gegebenen Datensatz von N Instanzen nur eine Instanz als Validierungsdatensatz (-instanz) ausgelassen wird und beim Training $N-1$ Instanzen genutzt werden. Wir erhalten dann N separate Paare, indem

LEAVE-ONE-OUT

bei jeder Iteration eine andere Instanz ausgelassen wird. Dieses Vorgehen wird typischerweise in Applikationen wie der medizinischen Diagnostik verwendet, bei denen es schwer ist, zugeordnete Daten zu finden. Der Leave-One-Out-Fall gestattet keine Stratifizierung.

Dank der gesunkenen Hardwarekosten ist es in den vergangenen Jahren auch möglich geworden, mehrere Durchläufe der K-fachen Kreuzvalidierung zu machen, beispielsweise 10×10-fach, und ein Mittel über Mittel zu verwenden, um vertrauenswürdigere Fehlerschätzungen zu bekommen (Bouckaert 2003).

19.6.2 5×2-Kreuzvalidierung

5×2-Kreuzvalidierung

Dietterich (1998) entwickelte die 5×2-*Kreuzvalidierung*, welche Trainings- und Validierungsdatensätze gleicher Größe benutzt. Wir zerlegen den Datensatz $\mathcal{X}$ zufällig in zwei Teile: $\mathcal{X}_1^{(1)}$ und $\mathcal{X}_1^{(2)}$, wodurch wir unser erstes Paar von Trainings- und Validierungsdaten erhalten: $\mathcal{T}_1 = \mathcal{X}_1^{(1)}$ und $\mathcal{V}_1 = \mathcal{X}_1^{(2)}$. Dann vertauschen wir die Rollen der beiden Hälften und erhalten das zweite Paar: $\mathcal{T}_2 = \mathcal{X}_1^{(2)}$ und $\mathcal{V}_2 = \mathcal{X}_1^{(1)}$. Dies ist der erste Durchgang; $\mathcal{X}_i^{(j)}$ bezeichnet die Hälfte j von Durchgang i.

Um zum zweiten Durchgang zu gelangen, mischen wir $\mathcal{X}$ zufällig und zerlegen diese neue Menge in zwei Teile, $\mathcal{X}_2^{(1)}$ und $\mathcal{X}_2^{(2)}$. Das kann durch die zufällige Ziehung dieser aus $\mathcal{X}$ ohne Zurücklegen geschehen, also $\mathcal{X}_1^{(1)} \cup \mathcal{X}_1^{(2)} = \mathcal{X}_2^{(1)} \cup \mathcal{X}_2^{(2)} = \mathcal{X}$. Wir vertauschen dann diese zwei Hälften, um ein weiteres Paar zu erhalten. Wir tun dies für drei weitere Durchgänge, und weil wir mit jedem Durchgang zwei Paare erhalten, ergeben sich bei fünf Durchgängen zehn Trainings- und Validierungsdatensätze:

$$
\begin{array}{ll}
\mathcal{T}_1 = \mathcal{X}_1^{(1)} & \mathcal{V}_1 = \mathcal{X}_1^{(2)} \\
\mathcal{T}_2 = \mathcal{X}_1^{(2)} & \mathcal{V}_2 = \mathcal{X}_1^{(1)} \\
\mathcal{T}_3 = \mathcal{X}_2^{(1)} & \mathcal{V}_3 = \mathcal{X}_2^{(2)} \\
\mathcal{T}_4 = \mathcal{X}_2^{(2)} & \mathcal{V}_4 = \mathcal{X}_2^{(1)} \\
\quad \vdots & \\
\mathcal{T}_9 = \mathcal{X}_5^{(1)} & \mathcal{V}_9 = \mathcal{X}_5^{(2)} \\
\mathcal{T}_{10} = \mathcal{X}_5^{(2)} & \mathcal{V}_{10} = \mathcal{X}_5^{(1)} \, .
\end{array}
$$

Natürlich können wir das Ganze auch für mehr als fünf Durchgänge durchführen und mehr Trainings-/Validierungsdatensätze erzeugen, doch Dietterich (1998) weist darauf hin, dass die Datensätze sich nach fünf Durchgängen viele Instanzen teilen und sich so stark überschneiden, dass die von diesen Sätzen berechneten Statistiken, also die Validierungsfehlerraten, zu stark voneinander abhängen und keinerlei neue Informationen beitragen. Selbst bei fünf Durchgängen überlappen sich die Mengen und

die Statistiken sind abhängig, jedoch können wir dies bei bis zu fünf Durchgängen noch verkraften. Andererseits, wenn wir weniger als fünf Durchgänge nutzen, dann erhalten wir weniger Daten (weniger als zehn) und haben keine ausreichend große Stichprobe, um daraus eine Wahrscheinlichkeitsfunktion zuverlässig zu schätzen anhand derer wir unsere Hypothese testen wollen.

19.6.3 Bootstrapping

BOOTSTRAP

Um multiple Stichproben aus einer einzelnen Stichprobe zu generieren, besteht eine Alternative zur Kreuzvalidierung in der *Bootstrap-Methode*, die neue Stichproben durch Ziehen von Instanzen aus der Originalstichprobe *mit* Zurücklegen generiert. Wir haben das Bootstrapping bereits in Abschnitt 17.6 kennengelernt, wo es verwendet wurde, um beim Bagging Trainingsmengen für verschiedene Lerner zu generieren. Die Bootstrap-Stichproben können sich mehr überschneiden als Kreuzvalidierungsstichproben, und somit sind ihre Schätzungen abhängiger; jedoch wird diese Methode als die beste für kleine Datensätze erachtet.

Beim Bootstrap ziehen wir stichprobenartig N Instanzen aus einer Datenmenge der Größe N mit Zurücklegen. Wenn wir einmal validieren, werden die Originaldaten als Validierungsmenge verwendet; andernfalls können wir mehrfach Trainings-/Validierungsdatensätze nach dieser Methode ziehen. Die Wahrscheinlichkeit, dass wir eine Instanz ziehen, ist $1/N$; die Wahrscheinlichkeit, dass wir sie nicht ziehen, ist $1 - 1/N$. Die Wahrscheinlichkeit, dass wir sie nicht nach N Ziehungen ziehen, ist

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0{,}368\,.$$

Das bedeutet, dass die Trainingsdaten etwa 63,2% der Instanzen enthalten; d. h., das System wird auf 36,8% der Daten nicht trainiert und die Fehlerschätzung wird pessimistisch sein. Die Lösung besteht darin, den Prozess viele Male zu wiederholen und den Durchschnitt zu bilden.

19.7 Leistungsmessung für Klassifikatoren

Für die Klassifikation und besonders für Zweiklassenprobleme ist eine Vielzahl von Maßen vorgeschlagen wurden. Es gibt vier mögliche Fälle, die in Tabelle 19.1 zusammengefasst sind. Wenn für eine positive Instanz die Vorhersage ebenfalls positiv ist, haben wir eine *richtig-positive* Entscheidung; ist die Vorhersage für eine positive Instanz negativ, ist die Entscheidung *falsch-negativ.* Für eine negative Instanz ist entsprechend eine negative Vorhersage *richtig-negativ,* und wir haben eine *falsch-negative* Entscheidung, wenn eine negative Instanz als positiv vorhergesagt wird.

Tabelle 19.1: Konfusionsmatrix für zwei Klassen

	vorhergesagte Klasse		
richtige Klasse	positiv	negativ	gesamt
positiv	rp: richtig-positiv	fn: falsch-negativ	p
negativ	fp: falsch-positiv	rn: richtig-negativ	n
gesamt	p'	n'	N

Bei manchen Zweiklassenproblemen machen wir einen Unterschied zwischen den beiden Klassen und folglich zwischen den beiden Arten von Fehlern, d. h. den falsch-positiven und den falsch-negativen Entscheidungen. In Tabelle 19.2 sind verschiedene Maße angegeben, die in den unterschiedlichen Situationen geeignet sind. Betrachten wir etwa eine Authentifikation, bei der die Nutzer sich über ihre Stimme einloggen. Falsch-positiv bedeutet in diesem Fall, dass sich ein Angreifer fälschlicherweise einloggen konnte, und falsch-negativ bedeutet, dass es ein berechtigter Nutzer abgewiesen wurde. Es ist offensichtlich, dass diese beiden Arten von Fehlern nicht gleich schlimm sind – der erste ist viel schlimmer. Die richtig-positiv-Rate (*rp-Rate*) wird auch *Trefferrate* genannt, und sie misst den Anteil der berechtigten Nutzer, die sich authentifizieren. Die falsch-positiv-Rate (*fp-Rate*), auch *Fehlalarm-Rate* genannt, ist der Anteil der Angreifer, die fälschlicherweise akzeptiert werden.

Tabelle 19.2: Maße für die Leistungsfähigkeit bei einem Zweiklassenproblem

Bezeichnung	Formel
Fehler	$(fp + fn)/N$
Genauigkeit	$(rp + rn)/N = 1 - \text{Fehler}$
rp-Rate	rp/p
fp-Rate	fp/n
Präzision	rp/p'
Trefferquote	$rp/p = \text{rp-Rate}$
Sensitivität	$rp/p = \text{rp-Rate}$
Spezifität	$rn/n = 1 - \text{fp-Rate}$

Angenommen, das System gibt $\hat{P}(C_1|x)$ zurück, die Wahrscheinlichkeit der positiven Klasse, und für die negative Klasse haben wir $\hat{P}(C_2|x) = 1 - \hat{P}(C_1|x)$, und wir wählen positiv, falls $\hat{P}(C_1|x) > \theta$. Wenn θ nahe bei 1 liegt, werden wir kaum die positive Klasse wählen, d. h., wir haben kaum

falsch-positive Entscheidungen, aber auch nur wenige richtig-positive. Wenn wir θ kleiner machen und damit die Zahl der richtig-positiven Entscheidungen erhöhen, steigt auch das Risiko für falsch-positive Entscheidungen.

ROC-Kurve

Wir können für unterschiedliche Werte von θ eine Reihe von Wertepaaren (rp-Rate, fp-Rate) erhalten, und indem wir diese verbinden, gelangen wir zur *Receiver Operating Characteristics* (ROC-Kurve), die in Abbildung 19.3a gezeigt ist. Man beachte, dass unterschiedliche Werte von θ unterschiedlichen Verlustmatrizen für die beiden Fehlerarten entsprechen und dass man die ROC-Kurve auch so interpretieren kann, dass sie das Verhalten eines Klassifikators unter verschiedenen Verlustmatrizen widerspiegelt (Übung 1).

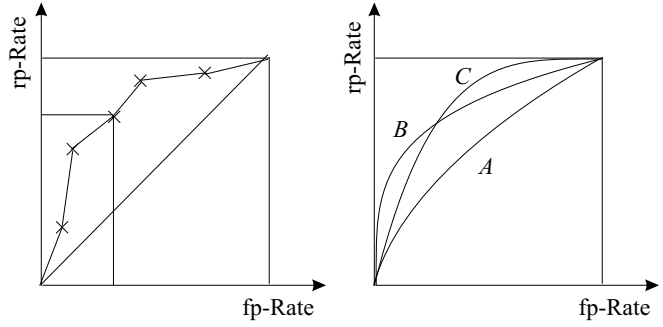


***Abb. 19.3*:** (a) Typische ROC-Kurve. Jeder Klassifikator besitzt einen Schwellwert, der es uns gestattet, entlang dieser Kurve zu fahren, und wir entscheiden für einen Punkt basierend auf der relativen Wichtigkeit von Treffern vs. Fehlalarmen, d. h. richtig-positiven und falsch-positiven Entscheidungen. Die Fläche unterhalb der ROC-Kurve wird AUC genannt. (b) Ein Klassifikator wird favorisiert, wenn seine ROC-Kurve näher an die linke obere Ecke reicht (eine größere AUC hat). B und C werden gegenüber A favorisiert; B und C werden unter verschiedenen Verlustmatrizen favorisiert.

Idealerweise hat ein Klassifikator eine rp-Rate von 1 und eine fp-Rate von 0, und folglich ist er umso besser, je näher seine ROC-Kurve an die linke obere Ecke reicht. Auf der Diagonale treffen wir genau so viele richtige Entscheidungen wie falsche, und das ist das Schlechteste, was man tun kann (denn jeder Klassifikator, der unterhalb der Diagonale liegt, lässt sich dadurch verbessern, dass man alle Entscheidungen umkehrt). Für zwei gegebene Klassifikatoren können wir sagen, dass der eine besser ist als der andere, falls seine ROC-Kurve über der des anderen liegt; falls sich die beiden Kurven schneiden, können wir sagen dass die beiden Klassifikatoren besser unter verschiedenen Verlustbedingungen sind (siehe Abbildung 19.3b).

Die ROC-Kurve erlaubt eine visuelle Analyse. Wenn wir die Information auf eine einzelne Zahl reduzieren wollen, dann können wir das tun, indem

wir die *Fläche unter der Kurve* (auch *AUC,* von engl. area under the curve) berechnen. Ein Klassifikator hat idealerweise eine AUC von 1, und indem wir AUC-Werte von unterschiedlichen Klassifikatoren vergleichen, können wir die allgemeine Leistungsfähigkeit (gemittelt über verschiedene Verlustbedingungen) einschätzen.

AUC

Beim *Information Retrieval* liegt uns eine Datenbasis mit vielen Datensätzen vor und wir stellen eine Suchanfrage, wobei wir eventuell bestimmte Keywords verwenden. Das System (im Wesentlichen ein Zweiklassen-Klassifikator) wird uns daraufhin eine Reihe von Datensätzen ausgeben. In der Datenbasis existieren relevante Datensätze, und für eine Suchanfrage wird das System einige davon auffinden und ausgeben (richtig-positiv), aber vermutlich nicht alle (falsch-negativ). Es kann auch fälschlicherweise Datensätze ausgeben, die nicht relevant sind (falsch-positiv). Die Menge der relevanten und aufgefundenen Datensätze kann durch ein Venn-Diagramm visualisiert werden, wie es in Abbildung 19.4a zu sehen ist. Die *Präzision* ist die Anzahl der aufgefundenen und relevanten Datensätze, geteilt durch die Gesamtzahl der aufgefundenen Datensätze. Wenn die Präzision 1 ist, sind alle aufgefundenen Datensätze relevant, aber es kann in der Datenbasis noch weitere relevante Datensätze geben, die nicht aufgefunden wurden. Die *Trefferquote* ist die Anzahl der aufgefundenen und relevanten Datensätze, geteilt durch die Gesamtzahl der relevanten Datensätze. Wenn die Trefferquote 1 ist, werden zwar alle relevanten Datensätze aufgefunden, doch können unter den aufgefundenen Datensätzen auch irrelevante sein (siehe Abbildung 19.4c). Wie bei der ROC-Kurve kann man für verschiedene Schwellwerte eine Kurve für die Präzision und die Trefferquote zeichnen.

Information Retrieval

Präzision

Trefferquote

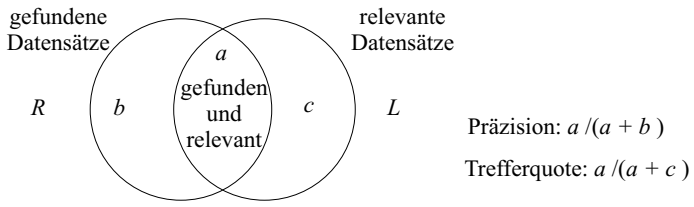
Eine andere Perspektive, aber die gleiche Zielsetzung haben die Begriffe *Sensitivität* und *Spezifität.* Die Sensitivität ist das gleiche wie die rp-Rate und die Trefferquote. Die Spezifität misst, wie gut die negativen Instanzen detektiert werden; sie ist definiert als das Verhältnis von richtig-negativen Entscheidungen zur Gesamtzahl der negativen Instanzen, und dieser Quotient ist 1 minus der Fehlalarmrate. Auch die Beziehung zwischen Sensitivität und Spezifität für verschiedene Schwellwerte kann durch eine Kurve dargestellt werden.

Sensitivität

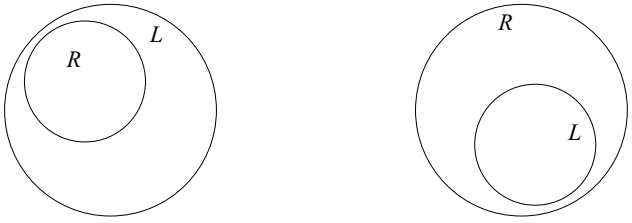
Spezifität

Wenn es $K > 2$ Klassen gibt und wir einen 0/1-Fehler verwenden, erweist sich die *Klassenkonfusionsmatrix* als nützlich. Es handelt sich um eine $K \times K$ Matrix, bei der ein Eintrag (i, j) die Anzahl an Instanzen enthält, die zu $\mathcal{C}_i$ gehören, aber $\mathcal{C}_j$ zugewiesen wurden. Idealerweise sollten alle Nichtdiagonalelemente gleich 0 sein, was bedeutet, dass es keine Fehlklassifikationen gibt. Die Klassenkonfusionsmatrix erlaubt es uns, genau hervorzuheben, welche Arten von Fehlklassifikationen auftreten, also ob es zwei Klassen gibt, die besonders häufig verwechselt werden. Alternativ können wir K separate Zweiklassenprobleme definieren, die jeweils eine Klasse von den anderen $K - 1$ separieren.

Klassenkonfusionsmatrix



(a) Präzision und Trefferquote



(b) Präzision = 1 (c) Trefferquote = 1

***Abb. 19.4*:** (a) Definition von Präzision und Trefferquote mithilfe eines Venn-Diagramms. (b) Die Präzision ist 1; alle aufgefundenen Datensätze sind relevant, doch es kann noch weitere Datensätze geben, die nicht aufgefunden wurden. (c) Die Trefferquote ist 1; alle relevanten Datensätze wurden aufgefunden, doch unter den aufgefundenen können sich auch irrelevante befinden.

19.8 Intervallschätzung

INTERVALL-SCHÄTZUNG

Rekapitulieren wir nun einmal kurz die *Intervallschätzung*, die wir beim Testen von Hypothesen anwenden werden. Eine Punktschätzfunktion, zum Beispiel der Maximum-Likelihood-Schätzer, spezifiziert einen Wert für einen Parameter θ. Bei der Intervallschätzung spezifizieren wir ein Intervall, innerhalb dessen θ mit einem gewissen Grad an Konfidenz liegt. Um so einen Intervallschätzer zu erstellen, machen wir Gebrauch von der Wahrscheinlichkeitsverteilung des Punktschätzers.

Sagen wir zum Beispiel, wir versuchen den Mittelwert μ einer normalverteilten Dichte aus einer Stichprobe $\mathcal{X} = \{x^t\}_{t=1}^N$ zu schätzen. $m = \sum_t x^t/N$ ist der Stichprobendurchschnitt und der Punktschätzer bezüglich des Mittelwertes. m ist die Summe der normalverteilten Dichten und ist somit auch selbst normalverteilt, $m \sim \mathcal{N}(\mu, \sigma^2/N)$. Wir definieren die Statistik mit einer *Standardnormalverteilung*:

STANDARDNORMAL-VERTEILUNG

$$\frac{(m-\mu)}{\sigma/\sqrt{N}} \sim \mathcal{Z} \,. \tag{19.1}$$

Wir wissen, dass 95 Prozent von $\mathcal{Z}$ in $(-1{,}96, 1{,}96)$ liegen, nämlich $P\{-1{,}96 < \mathcal{Z} < 1{,}96\} = 0{,}95$, und können notieren (siehe Abbil-

dung 19.5):

$$P\left\{-1{,}96 < \sqrt{N}\frac{(m-\mu)}{\sigma} < 1{,}96\right\} = 0{,}95$$

oder äquivalent dazu:

$$P\left\{m - 1{,}96\frac{\sigma}{\sqrt{N}} < \mu < m + 1{,}96\frac{\sigma}{\sqrt{N}}\right\} = 0{,}95\,.$$

Das heißt also, dass μ „mit 95-prozentiger Konfidenz“ innerhalb von $1{,}96\sigma/\sqrt{N}$ Einheiten des Stichprobendurchschnitts liegen wird. Hierbei handelt es sich um ein *zweiseitiges Konfidenzintervall*. Mit 99-prozentiger Konfidenz wird μ innerhalb von $(m - 2{,}58\sigma/\sqrt{N}, m + 2{,}58\sigma/\sqrt{N})$ liegen; das heißt, wollen wir mehr Konfidenz, dann vergrößert sich das Intervall. Das Intervall wird kleiner, wenn N, die Stichprobengröße, sich erhöht.

ZWEISEITIGES KONFIDENZINTERVALL

Das kann für jede beliebige geforderte Konfidenz wie folgt verallgemeinert werden: es sei z_α so, dass

$$P\{\mathcal{Z} > z_\alpha\} = \alpha,\ 0 < \alpha < 1\,.$$

Da $\mathcal{Z}$ symmetrisch um den Mittelwert herum liegt, ist $z_{1-\alpha/2} = -z_{\alpha/2}$ und $P\{X < -z_{\alpha/2}\} = P\{X > z_{\alpha/2}\} = \alpha/2$. Somit erhalten wir für eine beliebig spezifizierte Konfidenzstufe $1 - \alpha$

$$P\left\{-z_{\alpha/2} < \mathcal{Z} < z_{\alpha/2}\right\} = 1 - \alpha$$

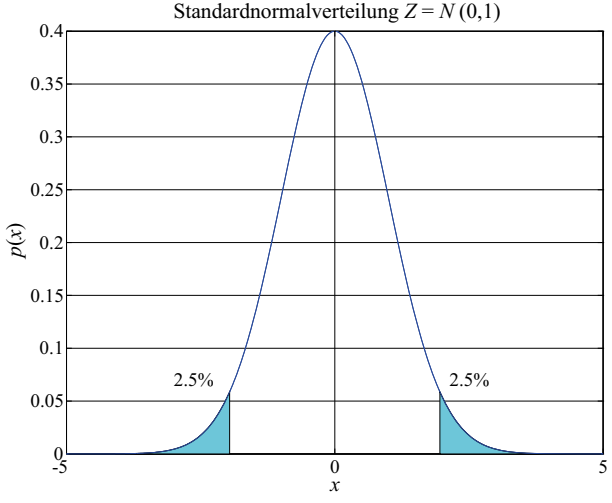


Abb. 19.5: 95% der Standardnormalverteilung liegen zwischen $-1{,}96$ und $1{,}96$.

und

$$P\left\{-z_{\alpha/2} < \sqrt{N}\frac{(m-\mu)}{\sigma} < z_{\alpha/2}\right\} = 1-\alpha$$

oder

$$P\left\{m - z_{\alpha/2}\frac{\sigma}{\sqrt{N}} < \mu < m + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right\} = 1-\alpha\,. \tag{19.2}$$

Somit kann ein $100(1-\alpha)$-prozentiges zweiseitiges Konfidenzintervall für μ für jedes beliebige α berechnet werden.

Auf ähnliche Weise erhalten wir mit dem Wissen, dass $P\{\mathcal{Z} < 1{,}64\} = 0{,}95$ (siehe Abbildung 19.6),

$$P\left\{\sqrt{N}\frac{(m-\mu)}{\sigma} < 1{,}64\right\} = 0{,}95$$

oder

$$P\left\{m - 1{,}64\frac{\sigma}{\sqrt{N}} < \mu\right\} = 0{,}95\,,$$

EINSEITIGES KONFIDENZINTERVALL

und $(m - 1{,}64\sigma/\sqrt{N}, \infty)$ ist ein 95-prozentiges *einseitiges oberes Konfidenzintervall* für μ, welches eine untere Schranke definiert. Verallgemeinernd kann ein $100(1-\alpha)$-prozentiges einseitiges Konfidenzintervall für μ berechnet werden aus

$$P\left\{m - z_{\alpha}\frac{\sigma}{\sqrt{N}} < \mu\right\} = 1-\alpha\,. \tag{19.3}$$

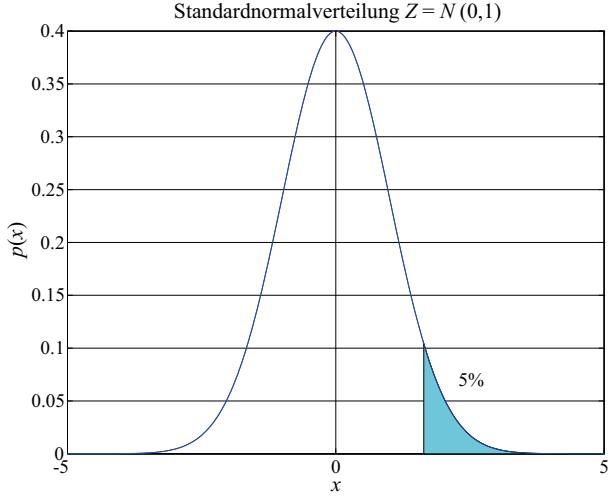


Abb. 19.6: 95% der Standardnormalverteilung liegen vor 1,64.

Ähnlich kann auch das einseitige untere Konfidenzintervall berechnet werden, das eine obere Schranke festlegt.

Bei den vorhergehenden Intervallen haben wir σ genutzt; das heißt, wir nahmen an, dass die Varianz bekannt ist. Ist sie das nicht, dann kann man die Stichprobenvarianz

$$S^2 = \sum_t (x^t - m)^2/(N-1)$$

statt σ^2 einsetzen. Wir wissen, wenn $x^t \sim \mathcal{N}(\mu, \sigma^2)$, dann unterliegt $(N-1)S^2/\sigma^2$ einer Chi-Quadrat-Verteilung mit $N-1$ Freiheitsgraden. Wir wissen auch, dass m und S^2 unabhängig sind. Dann ist $\sqrt{N}(m-\mu)/S$ t-verteilt mit $N-1$ Freiheitsgraden (Abschnitt A.3.7), geschrieben als:

$$\frac{\sqrt{N}(m-\mu)}{S} \sim t_{N-1} \,. \tag{19.4}$$

Somit können wir für jedes beliebige $\alpha \in (0, 1/2)$ ein Intervall definieren, indem wir die durch die *t-Verteilung* spezifizierten Werte statt das standardnormalverteilte $\mathcal{Z}$ nutzen: *t*-Verteilung

$$P\left\{t_{1-\alpha/2,N-1} < \sqrt{N}\frac{(m-\mu)}{S} < t_{\alpha/2,N-1}\right\} = 1-\alpha \,,$$

oder wir nutzen aus, dass $t_{1-\alpha/2,N-1} = -t_{\alpha/2,N-1}$:

$$P\left\{m - t_{\alpha/2,N-1}\frac{S}{\sqrt{N}} < \mu < m + t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}\right\} = 1-\alpha \,.$$

Auf vergleichbare Weise können einseitige Konfidenzintervalle definiert werden. Die t-Verteilung hat eine größere Streuung (längere Schweife) als die Standardnormalverteilung, und im Allgemeinen ist das durch t gegebene Intervall größer; das war zu erwarten, da zusätzliche Ungewissheit aufgrund der unbekannten Varianz entsteht.

19.9 Hypothesenprüfung

Statt einige Parameter explizit zu schätzen, möchten wir in gewissen Applikationen möglicherweise die Stichprobe nutzen, um eine bestimmte die Parameter betreffende Hypothese zu prüfen. Statt beispielsweise den Mittelwert zu schätzen, wollen wir möglicherweise testen, ob der Mittelwert kleiner als 0,02 ist. Ist die Zufallsstichprobe mit der betrachteten Hypothese konsistent, dann entscheiden wir uns für die „Nichtablehnung“ der Hypothese; andernfalls entscheiden wir uns für „Ablehnung“. Wenn wir so eine Entscheidung treffen, behaupten wir nicht, dass die Hypothese wahr oder falsch ist, sondern nur, dass die Stichprobendaten mit ihr bis

zu einem gewissen Grad an Konfidenz im Einklang stehen bzw. nicht im Einklang stehen.

HYPOTHESEN-PRÜFUNG

Bei der *Hypothesenprüfung* ist der Ansatz der folgende: wir definieren eine Statistik, die einer gewissen Verteilung folgt, wenn die Hypothese zutrifft. Wenn die anhand der Stichprobe berechnete Statistik eine ausreichend hohe Wahrscheinlichkeit dafür hat, aus dieser Verteilung gezogen worden zu sein, dann bestätigen wir die Hypothese; andernfalls lehnen wir sie ab.

Sagen wir, wir haben eine Stichprobe aus einer Normalverteilung mit unbekanntem Mittelwert μ und bekannter Varianz σ^2, und wir wollen eine spezifische Hypothese über μ prüfen, zum Beispiel, ob Gleichheit mit einer spezifizierten Konstante μ_0 besteht. Sie wird durch H_0 dargestellt und man spricht von der *Nullhypothese*

NULLHYPOTHESE

$$H_0 : \mu = \mu_0$$

im Gegensatz zur Alternativhypothese

$$H_1 : \mu \neq \mu_0 \, .$$

m ist die Punktschätzung von μ, und es ist sinnvoll, H_0 abzulehnen, wenn m zu weit von μ_0 entfernt liegt. Hier kommt die Intervallschätzung ins Spiel. Wir entscheiden uns für die Nichtablehnung der Hypothese mit einem *Signifikanzniveau* von α, wenn μ_0 im $100(1-\alpha)$-prozentigen Konfidenzintervall liegt, sprich, H_0 wird nicht abgelehnt, wenn

SIGNIFIKANZNIVEAU

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-z_{\alpha/2}, z_{\alpha/2}) \, . \tag{19.5}$$

Wir lehnen die Nullhypothese ab, wenn es außerhalb des Intervalls liegt (egal ob links oder rechts). Dies ist ein *zweiseitiger Test*.

ZWEISEITIGER TEST

***Tabelle 19.3*:** Fehler 1. Art, Fehler 2. Art und Trennschärfe eines Tests.

	Entscheidung	
Wahrheitswert	Nichtablehnung	Ablehnung
wahr	korrekt	Fehler 1. Art
falsch	Fehler 2. Art	korrekt

Wenn wir die Hypothese ablehnen, sie aber korrekt ist, dann handelt es sich um einen *Fehler 1. Art*; somit definiert der vor dem Test festgelegte Wert für α, inwieweit wir den Fehler 1. Art tolerieren können, wobei typische Werte hierfür $\alpha = 0{,}1, 0{,}05, 0{,}01$ sind (siehe Tabelle 19.3). Ein *Fehler 2. Art* entsteht durch Nichtablehnung der Nullhypothese, wenn der wahre Mittelwert μ ungleich μ_0 ist. Die Wahrscheinlichkeit für die Nichtablehnung von H_0, wenn der wahre Mittelwert μ ist, ist eine Funktion

FEHLER 1. ART

FEHLER 2. ART

von μ und gegeben durch

$$\beta(\mu) = P_\mu \left\{ -z_{\alpha/2} \leq \frac{m - \mu_0}{\sigma/\sqrt{N}} \leq z_{\alpha/2} \right\} . \tag{19.6}$$

$1 - \beta(\mu)$ nennt man die *Trennschärfe* des Tests und sie ist gleich der Wahrscheinlichkeit für eine Widerlegung der Hypothese, wenn μ der wahre Wert ist. Die Wahrscheinlichkeit für Fehler 1. Art wächst, wenn μ und μ_0 näher beieinander liegen, und wir können berechnen, wie groß eine Stichprobe sein muss, damit wir in der Lage sind, eine Differenz $\delta = |\mu - \mu_0|$ mit ausreichender Trennschärfe zu detektieren.

TRENNSCHÄRFE

Man kann auch einen *einseitigen Test* der Form

EINSEITIGER TEST

$$H_0 : \mu \leq \mu_0 \text{ vs. } H_1 : \mu > \mu_0$$

im Gegensatz zum zweiseitigen Test nutzen, wenn die Alternativhypothese $\mu \neq \mu_0$ ist. Der einseitige Test mit Signifikanzniveau α definiert das an einer Seite begrenzte $100(1 - \alpha)$-Konfidenzintervall, in dem m liegen sollte, damit die Hypothese nicht abgelehnt wird. Wir lehnen nicht ab, falls

$$\frac{\sqrt{N}}{\sigma}(m - \mu_0) \in (-\infty, z_\alpha) ; \tag{19.7}$$

andernfalls lehnen wir ab. Man beachte, dass die Nullhypothese H_0 auch Gleichheit zulässt, was bedeutet, dass wir Ordnungsinformationen nur im Falle der Ablehnung bekommen. Damit ist klar, welchen der beiden einseitigen Tests wir verwenden sollten. Was immer unsere Behauptung ist, sollte in H_1 sein, weil dann die Ablehnung durch den Test unsere Behauptung stützen wird.

Ist die Varianz unbekannt, dann nutzen wir, wie schon bei den Intervallschätzungen, die Stichprobenvarianz statt der Populationsvarianz sowie die Tatsache, dass

$$\frac{\sqrt{N}(m - \mu_0)}{S} \sim t_{N-1} . \tag{19.8}$$

Zum Beispiel entscheiden wir auf Nichtablehnung bei $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$ mit einem Signifikanzniveau α, falls

$$\frac{\sqrt{N}(m - \mu_0)}{S} \in (-t_{\alpha/2,N-1}, t_{\alpha/2,N-1}) . \tag{19.9}$$

Dies ist als der zweiseitige *t-Test* bekannt. Ein einseitiger t-Test kann auf ähnliche Weise definiert werden.

t-TEST

19.10 Bewertung der Leistungsfähigkeit von Klassifikationsalgorithmen

Nun, da wir die Hypothesenprüfung besprochen haben, wird es Zeit, dass wir uns anschauen, wie sie beim Testen von Fehlerraten eingesetzt wird. Wir werden den Fall des Klassifikationsfehlers diskutieren, doch die gleiche Methodik kann auch für den quadratischen Fehler bei der Regression, für Log-Likelihoods beim nichtüberwachten Lernen oder für die erwartete Belohnung beim bestärkenden Lernen angewendet werden, solange wir nur die geeignete parametrische Form der Stichprobenverteilung aufschreiben können. Wir werden auch nichtparametrische Tests diskutieren für den Fall, dass keine solche parametrische Form gefunden werden kann.

Wir beginnen mit der Bewertung von Fehlerraten und diskutieren im folgenden Abschnitt den Vergleich von Fehlerraten.

19.10.1 Binomialtest

Beginnen wir mit dem Fall, dass wir eine einzelne Trainingsmenge $\mathcal{T}$ und eine einzelne Validierungsmenge $\mathcal{V}$ haben. Wir trainieren unseren Klassifikator an $\mathcal{T}$ und testen ihn an $\mathcal{V}$. Wir bezeichnen mit p die Wahrscheinlichkeit, dass der Klassifikator einen Fehlklassifikationsfehler verursacht. Wir kennen p nicht, sondern es handelt sich um den Wert, hinsichtlich dessen wir eine Hypothese schätzen oder prüfen wollen. Sei x^t die Korrektheit der Entscheidung des Klassifikators bei einer Instanz mit Index t aus dem Validierungsdatensatz $\mathcal{V}$. Dann ist x^t eine 0/1-verteilte Zufallsvariable, d. h., es hat eine Bernoulli-Verteilung. x^t nimmt den Wert 1 an, wenn der Klassifikator einen Fehler begeht und den Wert 0, wenn der Klassifikator korrekt ist. Die binomialverteilte Zufallsvariable X bezeichnet die Geamtzahl der Fehler:

$$X = \sum_{t=1}^{N} x^t \,.$$

Wir würden gern testen, ob die Fehlerwahrscheinlichkeit p kleiner oder gleich eines von uns definierten Wertes p_0 ist:

$$H_0 : p \leq p_0 \text{ vs. } H_1 : p > p_0 \,.$$

Ist die Fehlerwahrscheinlichkeit gleich p, dann ist die Wahrscheinlichkeit, dass der Klassifikator j Fehler aus N verursacht, gleich

$$P\{X = j\} = \left(\begin{array}{c} N \\ j \end{array} \right) p^j (1-p)^{N-j} \,.$$

Es ist sinnvoll, $p \leq p_0$ in einem solchen Fall abzulehnen, wenn die Wahrscheinlichkeit, $X = e$ oder mehr Fehler zu sehen, sehr klein ist. Das

heißt, der *Binomialtest* lehnt die Hypothese ab, falls

Binomialtest

$$P\{X \geq e\} = \sum_{x=e}^{N} \binom{N}{x} {p_0}^x (1-p_0)^{N-x} < \alpha \,. \tag{19.10}$$

Dabei ist α die Signifikanz, die zum Beispiel 0,05 sein kann.

19.10.2 Test der approximierten Normalverteilung

Wenn p die Wahrscheinlichkeit des Fehlers ist, dann lautet unsere Schätzung $\hat{p} = X/N$. Es ist dann sinnvoll, die Nullhypothese abzulehnen, wenn $\hat{p}$ deutlich größer ist als p_0. Was dabei groß genug ist, wird durch die Stichprobenverteilung von $\hat{p}$ und die Signifikanz α bestimmt.

Weil X die Summe von unabhängigen Zufallsvariablen aus derselben Verteilung ist, besagt der zentrale Grenzwertsatz, dass X/N für große N näherungsweise normalverteilt ist mit dem Mittelwert p_0 und der Varianz $p_0(1-p_0)/N$. Dann ist

$$\frac{X/N - p_0}{\sqrt{p_0(1-p_0)/N}} \dot{\sim} \mathcal{Z} \,, \tag{19.11}$$

wobei $\dot{\sim}$ für „näherungsweise verteilt“ steht. Dann lehnt der *Test der approximierten Normalverteilung* die Nullhypothese ab (wobei Gleichung 19.7 verwendet wird), wenn dieser Wert für $X = e$ größer als z_α ist. $z_{0,05}$ ist 1,64. Dieser Test funktioniert gut, solange N nicht zu klein wird und p nicht zu dicht an 0 oder 1 liegt. Als Faustregel fordern wir $Np \geq 5$ und $N(1-p) \geq 5$.

Test der approximierten Normalverteilung

19.10.3 t-Test

Die zwei bereits diskutierten Tests nutzen einen einzelnen Validierungsdatensatz. Durchlaufen wir den Algorithmus K Mal, an K Paaren von Trainings-/Validierungsdaten, erhalten wir K Fehlerprozentsätze, $p_i, i = 1, \ldots, K$, an den K Validierungsdatensätzen. Sei x_i^t gleich 1, wenn der an $\mathcal{T}_i$ trainierte Klassifikator an Instanz t von $\mathcal{V}_i$ einen Fehlklassifikationsfehler verursacht; andernfalls ist x_i^t gleich 0. Dann gilt

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N} \,.$$

Es sei gegeben, dass

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K-1} \,,$$

dann wissen wir laut Gleichung 19.8, dass wir

$$\frac{\sqrt{K}(m - p_0)}{S} \sim t_{K-1} \tag{19.12}$$

erhalten und der t-Test lehnt die Nullhypothese ab, dass der Klassifikationsalgorithmus einen Fehlerprozentsatz von p_0 oder niedriger hat, und zwar mit Signifikanzniveau α, wenn dieser Wert kleiner oder gleich $t_{\alpha,K-1}$ ist. Typischerweise wird für K der Wert 10 oder 30 genutzt. $t_{0,05,9} = 1{,}83$ und $t_{0,05,29} = 1{,}70$.

19.11 Vergleich von zwei Klassifikationsalgorithmen

Wir wollen zwei gegebene Lernalgorithmen vergleichen und testen, ob sie Klassifikatoren konstruieren, die dieselbe erwartete Fehlerrate haben.

19.11.1 Der McNemarsche Test

Bei vorliegendem Trainings- und Validierungsdatensatz nutzen wir zwei Algorithmen, um zwei Klassifikatoren am Trainingsdatensatz zu trainieren und testen sie anschließend am Validierungsdatensatz und berechnen ihre Fehler. Eine *Kontingenztafel*, wie die hier dargestellte, ist eine Menge an natürlichen Zahlen in Matrixform, die Zähler oder Frequenzen darstellen:

KONTINGENZTAFEL

e_{00}: Anzahl der Beispiele die von beiden fehlklassifiziert wurden	e_{01}: Anzahl der Beispiele die durch 1 aber nicht durch 2 fehlklassifiziert wurden
e_{10}: Anzahl der Beispiele die durch 2 aber nicht durch 1 fehlklassifiziert wurden	e_{11}: Anzahl der Beispiele die durch beide korrekt klassifiziert wurden

Unter der Nullhypothese, dass die Klassifikationsalgorithmen dieselbe Fehlerrate besitzen, erwarten wir $e_{01} = e_{10}$ sowie, dass diese gleich $(e_{01} + e_{10})/2$ sind. Wir haben die Chi-Quadrat-Statistik mit einem Freiheitsgrad

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \mathcal{X}_1^2 \tag{19.13}$$

MCNEMARSCHER TEST

und der *McNemarsche Test* verifiziert die Hypothese, dass die zwei Klassifikationsalgorithmen dieselbe Fehlerrate haben, und zwar mit Signifikanzniveau α, wenn dieser Wert kleiner oder gleich $\mathcal{X}_{\alpha,1}^2$ ist. $\mathcal{X}_{0,05,1}^2 = 3{,}84$.

19.11.2 Gepaarter t-Test mit K-facher Kreuzvalidierung

Dieser Test nutzt K-fache Kreuzvalidierung, um K Paare an Trainings-/Validierungsdaten zu erhalten. Wir nutzen die zwei Klassifikationsalgorithmen, um an den Trainingsdatensätzen $\mathcal{T}_i, i = 1, \ldots, K$ zu trainieren

und testen mit den Validierungsdatensätzen $\mathcal{V}_i$. Die Fehlerprozentsätze der Klassifikatoren an den Validierungsmengen werden als p_i^1 und p_i^2 aufgezeichnet.

Wenn die zwei Klassifikationsalgorithmen dieselbe Fehlerrate haben, dann erwarten wir, dass sie denselben Mittelwert haben, oder äquivalent dazu, dass die Differenz zwischen ihren Mittelwerten gleich 0 ist. Die Differenz in den Fehlerraten am Durchgang i ist $p_i = p_i^1 - p_i^2$. Das ist ein *gepaarter Test,* worunter wir verstehen, dass beide Algorithmen für jedes i die gleichen Trainings- und Validierungsmengen sehen. Wird dies K Mal durchgeführt, dann erhalten wir eine Verteilung von p_i, die K Punkte beinhaltet. Gegeben, dass p_i^1 und p_i^2 beide (annähernd) normalverteilt sind, ist ihre Differenz p_i auch normalverteilt. Die Nullhypothese lautet, dass diese Verteilung einen Nullmittelwert aufweist:

GEPAARTER TEST

$$H_0 : \mu = 0 \text{ vs. } H_1 : \mu \neq 0 .$$

Wir definieren

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, \quad S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1} .$$

Unter der Nullhypothese, dass $\mu = 0$, haben wir eine Statistik, die t-verteilt mit $K - 1$ Freiheitsgraden ist:

$$\frac{\sqrt{K}(m - 0)}{S} = \frac{\sqrt{K} \cdot m}{S} \sim t_{K-1} . \tag{19.14}$$

Somit verifiziert der *K-fache kreuzvalidierte gepaarte t-Test* die Hypothese, dass zwei Klassifikationsalgorithmen dieselbe Fehlerrate haben, und zwar mit Signifikanzniveau α, wenn dieser Wert im Intervall $(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1})$ liegt. $t_{0,025,9} = 2{,}26$ und $t_{0,025,29} = 2{,}05$.

K-FACHER KREUZVALIDIERTER GEPAARTER t-TEST

Wenn wir testen wollen, ob der erste Algorithmus weniger Fehler macht als der zweite, brauchen wir eine einseitige Hypothese und verwenden einen einseitigen Test:

$$H_0 : \mu \geq 0 \text{ vs. } H_1 : \mu < 0 .$$

Wenn der Test ablehnt, wird unsere Behauptung, dass der erste Algorithmus signifikant weniger Fehler macht, gestützt.

19.11.3 Gepaarter t-Test mit 5×2 Kreuzvalidierung

Beim 5×2 kreuzvalidierten t-Test, der durch Dietterich (1998) entworfen wurde, führen wir fünf Replikationen von zweifacher Kreuzvalidierung

durch. Bei jeder Replikation wird der Datensatz in zwei gleichgroße Mengen zerlegt. $p_i^{(j)}$ ist die Differenz zwischen den Fehlerraten der zwei Klassifikatoren an Durchgang $j = 1, 2$ aus Replikation $i = 1, \ldots, 5$. Der Durchschnitt für Replikation i ist $\overline{p}_i = (p_i^{(1)} + p_i^{(2)})/2$ und die geschätzte Varianz ist $s_i^2 = (p_i^{(1)} - \overline{p}_i)^2 + (p_i^{(2)} - \overline{p}_i)^2$.

Unter der Nullhypothese, dass die zwei Klassifikationsalgorithmen dieselbe Fehlerrate haben, ist $p_i^{(j)}$ die Differenz von zwei identisch verteilten Zufallsvariablen, und wenn wir die Tatsache ignorieren, dass diese Proportionen nicht unabhängig sind, dann kann $p_i^{(j)}$ als ungefähr normalverteilt mit Nullmittelwert und unbekannter Varianz σ^2 behandelt werden. Dann ist $p_i^{(j)}/\sigma$ ungefähr standardnormalverteilt. Wenn wir annehmen, dass $p_i^{(1)}$ und $p_i^{(2)}$ unabhängige Normalverteilungen sind (was streng genommen nicht wahr ist, weil ihre Trainings- und Testdatensätze nicht unabhängig voneinander gezogen werden), dann hat s_i^2/σ^2 eine Chi-Quadrat-Verteilung mit einem Freiheitsgrad. Wenn angenommen wird, dass jedes s_i^2 unabhängig ist (was nicht wahr ist, da sie alle aus derselben Menge an verfügbaren Daten berechnet wurden), dann unterliegt ihre Summe einer Chi-Quadrat-Verteilung mit fünf Freiheitsgraden:

$$M = \frac{\sum_{i=1}^{5} s_i^2}{\sigma^2} \sim \mathcal{X}_5^2$$

und

$$t = \frac{p_1^{(1)}/\sigma}{\sqrt{M/5}} = \frac{p_1^{(1)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \sim t_5 \,, \tag{19.15}$$

5×2 KREUZVALIDIERTER GEPAARTER t-TEST

wodurch wir eine t-Statistik mit fünf Freiheitsgraden erhalten. Der 5×2 *kreuzvalidierte gepaarte t-Test* verifiziert die Hypothese, dass die zwei Klassifikationsalgorithmen dieselbe Fehlerrate haben, und zwar mit Signifikanzniveau α, falls dieser Wert im Intervall $(-t_{\alpha/2,5}, t_{\alpha/2,5})$ liegt. $t_{0{,}025,5} = 2{,}57$.

19.11.4 Gepaarter F-Test mit 5×2 Kreuzvalidierung

Wir bemerken, dass der Zähler in Gleichung 19.15, $p_1^{(1)}$, willkürlich ist; tatsächlich können zehn verschiedene Werte in den Zähler eingesetzt werden, nämlich $p_i^{(j)}$, $j = 1, 2$, $i = 1, \ldots, 5$, was zu zehn möglichen Statistiken führt:

$$t_i^{(j)} = \frac{p_i^{(j)}}{\sqrt{\sum_{i=1}^{5} s_i^2/5}} \,. \tag{19.16}$$

Alpaydın (1999) entwarf eine Erweiterung zum 5×2 kreuzvalidierten t-Test, welche die Ergebnisse von den zehn möglichen Statistiken kombiniert. Falls $p_i^{(j)}/\sigma \sim \mathcal{Z}$, dann $\left(p_i^{(j)}\right)^2/\sigma^2 \sim \mathcal{X}_1^2$ und ihre Summe ist eine Chi-Quadrat-Verteilung mit zehn Freiheitsgraden:

$$N = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{\sigma^2} \sim \mathcal{X}_{10}^2 \,.$$

Setzen wir dies in den Zähler aus Gleichung 19.15 ein, dann erhalten wir eine Statistik, welche das Verhältnis der zwei Chi-Quadrat-Verteilungen der Zufallsvariablen ist. Zwei solche Variablen geteilt durch ihre jeweiligen Freiheitsgrade sind F-verteilt mit zehn bzw. fünf Freiheitsgraden (Abschnitt A.3.8):

$$f = \frac{N/10}{M/5} = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5} s_i^2} \sim F_{10,5} \,. \tag{19.17}$$

Der 5×2 *kreuzvalidierte gepaarte F-Test* lehnt die Hypothese ab, dass die Klassifikationsalgorithmen dieselbe Fehlerrate haben, und zwar mit dem Signifikanzniveau α, falls dieser Wert größer als $F_{\alpha,10,5}$. $F_{0,05,10,5} = 4{,}74$ ist.

5×2 KREUZVALIDIERTER GEPAARTER F-TEST

19.12 Vergleich mehrerer Algorithmen: Varianzanalyse

In vielen Fällen haben wir mehr als zwei Algorithmen und wir wollen ihre erwarteten Fehler vergleichen. Wir haben L gegebene Algorithmen und trainieren diese an K Trainingsdatensätzen, induzieren K Klassifikatoren mit jedem Algorithmus, testen sie dann an K Validierungsmengen und zeichnen ihre Fehlerraten auf. Dies liefert uns L Gruppen mit K Werten. Das Problem besteht dann im Vergleich dieser L Stichproben auf statistisch signifikante Unterschiede hin. Dies ist ein Experiment mit einem einzelnen Faktor, der L Einstellmöglichkeiten hat (die Lernalgorithmen), und es gibt für jede Einstellung K Wiederholungen.

Bei der *Varianzanalyse* (auch ANOVA für engl. *analysis of variance*) betrachten wir L unabhängige Stichproben, jede von der Größe K, die aus normalverteilten Zufallsvariablen von unbekanntem Mittelwert μ_j und unbekannter gemeinsamer Varianz σ^2 zusammengesetzt sind:

VARIANZANALYSE

$$X_{ij} \sim \mathcal{N}(\mu_j, \sigma^2), j = 1, \ldots, L,\ i = 1, \ldots, K \,,$$

und uns interessiert die Prüfung der Hypothese H_0, dass alle Mittelwerte gleich sind:

$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L$ vs. $H_1 : \mu_r \neq \mu_s$ für mindestens ein Paar (r, s).

Der Vergleich von Fehlerraten von multiplen Klassifikationsalgorithmen passt in dieses Schema. Wir haben L Klassifikationsalgorithmen und uns liegen ihre Fehlerraten an K Validierungsdurchgängen vor. X_{ij} ist die Anzahl an Validierungsfehlern, die vom Klassifikator verursacht wurden, der durch den Klassifikationsalgorithmus j an Durchgang i trainiert wurde. Jedes X_{ij} ist binomial und ungefähr normalverteilt. Wird H_0 nicht abgelehnt, so finden wir keine signifikante Fehlerdifferenz zwischen den Fehlerraten der L Klassifikationsalgorithmen. Es handelt sich somit um eine Verallgemeinerung der Tests, die wir in Abschnitt 19.11 betrachtet haben und welche die Fehlerraten von zwei Klassifikationsalgorithmen verglichen. Die L Klassifikationsalgorithmen können sich unterscheiden oder verschiedene Hyperparameter nutzen, zum Beispiel die Anzahl an verborgenen Einheiten in einem mehrlagigen Perzeptron, die Anzahl an Nachbarn beim k-NN, und so weiter.

Der Ansatz bei der Varianzanalyse besteht darin, zwei Schätzer von σ^2 abzuleiten. Einer dieser beiden Schätzer wird unter der Annahme, dass H_0 zutrifft, hergeleitet. Der zweite Schätzer ist unabhängig von H_0 immer gültig. Die Varianzanalyse lehnt dann H_0 ab, also die Hypothese, dass die L Stichproben aus derselben Population entnommen wurden, wenn die zwei Schätzer sich signifikant unterscheiden.

Unser erster Schätzer für σ^2 ist nur gültig, wenn die Hypothese wahr ist, also wenn $\mu_j = \mu, j = 1, \ldots, L$. Falls $X_{ij} \sim \mathcal{N}(\mu, \sigma^2)$, dann ist der Gruppendurchschnitt

$$m_j = \sum_{i=1}^{K} \frac{X_{ij}}{K}$$

auch normalverteilt mit Mittelwert μ und Varianz σ^2/K. Ist die Hypothese wahr, dann sind $m_j, j = 1, \ldots, L$ hier L Instanzen, die aus $\mathcal{N}(\mu, \sigma^2/K)$ gezogen wurden. Somit sind *deren* Mittelwert und Varianz

$$m = \frac{\sum_{j=1}^{L} m_j}{L}, \quad S^2 = \frac{\sum_j (m_j - m)^2}{L - 1} .$$

Also ist ein Schätzer von σ^2 gleich $K \cdot S^2$, sprich

$$\hat{\sigma}^2 = K \sum_{j=1}^{L} \frac{(m_j - m)^2}{L - 1} . \tag{19.18}$$

Jedes der m_j ist normalverteilt und $(L-1)S^2/(\sigma^2/K)$ ist eine Chi-Quadrat-Verteilung mit $(L-1)$ Freiheitsgraden. Dann erhalten wir

$$\sum_j \frac{(m_j - m)^2}{\sigma^2/K} \sim \mathcal{X}^2_{L-1} . \tag{19.19}$$

Wir definieren SS_b, die Summe der Quadrate zwischen den Gruppen, als

$$SS_b \equiv K \sum_j (m_j - m)^2 .$$

Wenn also H_0 wahr ist, erhalten wir

$$\frac{SS_b}{\sigma^2} \sim \mathcal{X}^2_{L-1} . \tag{19.20}$$

Unser zweiter Schätzer von σ^2 ist der Durchschnitt der Gruppenvarianzen, S_j^2, definiert durch

$$S_j^2 = \frac{\sum_{i=1}^{K} (X_{ij} - m_j)^2}{K-1}$$

und ihr Durchschnitt ist

$$\hat{\sigma}^2 = \sum_{j=1}^{L} \frac{S_j^2}{L} = \sum_j \sum_i \frac{(X_{ij} - m_j)^2}{L(K-1)} . \tag{19.21}$$

Wir definieren SS_w, die Summe der Quadrate innerhalb der Gruppen:

$$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2 .$$

Wir erinnern uns, dass bei einer normalverteilten Stichprobe

$$(K-1)\frac{S_j^2}{\sigma^2} \sim \mathcal{X}^2_{K-1}$$

und dass die Summe der Chi-Quadrate auch ein Chi-Quadrat ist; wir erhalten daher

$$(K-1)\sum_{j=1}^{L} \frac{S_j^2}{\sigma^2} \sim \mathcal{X}^2_{L(K-1)} .$$

Somit ist

$$\frac{SS_w}{\sigma^2} \sim \mathcal{X}^2_{L(K-1)} . \tag{19.22}$$

Unsere Aufgabe besteht dann darin, zwei Varianzen auf Gleichheit zu prüfen, was wir tun können, indem wir testen, ob ihr Quotient nahe bei 1 liegt. Das Verhältnis von zwei unabhängigen Chi-Quadrat-verteilten

Zufallsvariablen geteilt durch ihre jeweiligen Freiheitsgrade ist eine F-verteilte Zufallsvariable, und somit erhalten wir, wenn H_0 wahr ist:

$$F_0 = \left(\frac{SS_b/\sigma^2}{L-1}\right) \Big/ \left(\frac{SS_w/\sigma^2}{L(K-1)}\right) = \frac{SS_b/(L-1)}{SS_w/(L(K-1))} \sim F_{L-1,L(K-1)} \,. \tag{19.23}$$

Für einen beliebigen gegebenen Signifikanzwert α wird die Hypothese abgelehnt, dass die L Klassifikationsalgorithmen dieselbe erwartete Fehlerrate haben, wenn diese Statistik größer ist als $F_{\alpha,L-1,L(K-1)}$.

Man beachte, dass wir ablehnen, wenn die beiden Schätzer signifikant voneinander abweichen. Wenn H_0 nicht richtig ist, dann wird die Varianz von m_j um m größer sein als es normalerweise der Fall wäre, wenn H_0 richtig wäre. Folglich wird der erste Schätzer $\hat{\sigma}_b^2$, wenn H_0 nicht richtig ist, σ^2 überschätzen, und das Verhältnis wird größer sein als 1. Für $\alpha = 0{,}05$, $L = 5$ und $K = 10$ ist $F_{0{,}05,4,45} = 2{,}6$. Wenn X_{ij} mit einer Varianz von σ^2 um m variiert, dann variieren die m_j, falls H_0 richtig ist, mit σ^2/K um m. Wenn es so scheint, als würden sie stärker variieren, dann sollte H_0 abgelehnt werden, weil die Abweichung von m_j gegenüber m zu groß ist, um sie durch ein konstantes additives Rauschen zu erklären.

Der Begriff Varianzanalyse erklärt sich aus dem Umstand, dass die Gesamtvariabilität innerhalb der Daten in ihre Komponenten zerlegt wird:

$$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2 \,. \tag{19.24}$$

SS_T geteilt durch seine Freiheitsgrade, nämlich $K \cdot (L-1)$ (es gibt $K \cdot L$ Datenpunkte und wir verlieren einen Freiheitsgrad, weil wir m festhalten), liefert uns die Stichprobenvarianz von X_{ij}. Man kann zeigen (Übung 5), dass die Gesamtsumme der Quadrate in eine Summe zwischen den Gruppen (SS_b für „between“) und eine Summe innerhalb der Gruppen (SS_w für „within“) aufgespalten werden kann:

$$SS_T = SS_b + SS_w \,. \tag{19.25}$$

Ergebnisse von ANOVA sind in Tabelle 19.4 in Form einer ANOVA-Tafel dargestellt. Dies ist die Standard-Einweg-Analyse der Varianz, bei der es einen einzigen Faktor, beispielsweise einen Lernalgorithmus, gibt. Wir können auch Experimente mit mehreren Faktoren betrachten, etwa den Fall, dass wir einen Faktor für die Klassifikationsalgorithmen und einen zweiten für die zuvor verwendeten Algorithmen zur Merkmalsextraktion haben. Letzteres wäre ein Zwei-Faktoren-Experiment mit Interaktion.

Wenn die Hypothese abgelehnt wird, wissen wir nur, dass es eine Differenz zwischen den L Gruppen gibt, wir wissen aber nicht wo. Um dies herauszufinden, führen wir *post-hoc-Tests* durch, d. h. eine Reihe zusätzlicher

POST-HOC-TESTS

Tabelle 19.4: ANOVA-Tafel für ein Modell mit einem einzelnen Faktor

Quelle	Summe der Quadrate	Freiheitsgrade	mittlere quadratische Abweichung	F_0
zwischen Gruppen	$SS_b \equiv K\sum_j (m_j - m)^2$	$L-1$	$MS_b = \frac{SS_b}{L-1}$	$\frac{MS_b}{MS_w}$
innerhalb v. Gruppen	$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$	$L(K-1)$	$MS_w = \frac{SS_w}{L(K-1)}$	s
gesamt	$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2$	$L \cdot K - 1$		

Tests, bei denen Teilmengen von Gruppen, beispielsweise Paare, involviert sind.

FISHER-TEST

Der *Fisher-Test* für die Differenz der kleinsten Quadrate vergleicht Gruppen paarweise. Für jede Gruppe haben wir $m_i \sim \mathcal{N}(\mu_i, \sigma_w^2 = MS_w/K)$ und $m_i - m_j \sim \mathcal{N}(\mu_i - \mu_j, 2\sigma_w^2)$. Unter der Nullhypothese $H_0 \colon \mu_i = \mu_j$ haben wir dann

$$t = \frac{m_i - m_j}{\sqrt{2}\sigma_w} \sim t_{L(K-1)} \,.$$

Wir lehnen H_0 ab und favorisieren die Alternativhypothese $H_1 : \mu_1 \neq \mu_2$, wenn $|t| > t_{\alpha/2, L(K-1)}$. Analog dazu können einseitige Tests definiert werden, um die paarweise Ordnung zu finden.

MULTIPLER VERGLEICH

Wenn wir eine Reihe von Tests durchführen, um eine Schlussfolgerung zu ziehen, sprechen wir von *multiplen Vergleichen.* Dabei erinnern wir uns, dass beim Testen von T Hypothesen, von denen jede das Signifikanzniveau α hat, die Wahrscheinlichkeit, dass mindestens eine Hypothese fälschlicherweise abgelehnt wird, höchstens $T\alpha$ ist. Beispielsweise ist die Wahrscheinlichkeit, dass sechs Konfidenzintervalle, von denen jedes mit 95-prozentigem individuellen Konfidenzintervall berechnet wurde, gleichzeitig korrekt sein werden, mindestens 70%. Um somit sicherzustellen, dass das allumfassende Konfidenzintervall mindestens $100(1-\alpha)$ ist, sollte jedes Konfidenzintervall auf $100(1 - \alpha/m)$ gesetzt werden. Dies wird als *Bonferroni-Korrektur* bezeichnet.

BONFERRONI-KORREKTUR

Manchmal kommt es vor, dass ANOVA ablehnt und keiner der paarweisen post-hoc-Tests einen signifikanten Unterschied findet. In diesem Fall ist unsere Schlussfolgerung die, dass es einen Unterschied zwischen den Mittelwerten gibt, wir jedoch mehr Daten benötigen, um die Ursache für den Unterschied zu lokalisieren.

Man beachte, dass der Hauptaufwand durch das Training und Testen von L Klassifikationsalgorithmen an K Trainings-/Validierungsdatensätzen entsteht. Ist dies einmal vollbracht und sind die Werte in einer

$K \times L$ Tabelle abgespeichert, dann ist die Berechnung der Varianzanalyse oder der paarweisen Vergleichsteststatistiken anhand dieser Werte relativ mühelos.

19.13 Vergleich über mehrere Datenmengen

Nehmen wir an, wir wollen zwei oder mehr Algorithmen auf mehreren Datenmengen anstatt nur auf einer vergleichen. Was diese Problemstellung verschieden von der bisher betrachteten macht, ist der Umstand, dass ein Algorithmus, der davon abhängt, wie gut sich seine induktive Verzerrung dem Problem anpasst, sich auf unterschiedlichen Datenmengen unterschiedlich verhalten wird. Von diesen Fehlerwerten auf unterschiedlichen Datenmengen kann man nicht sagen, dass sie um eine mittlere Genauigkeit normalverteilt sind. Das bedeutet, dass parametrische Tests, wie wir sie in den vorherigen Abschnitten diskutiert haben (basierend auf Binomialverteilung, die näherungsweise Normalverteilungen sind), nicht mehr anwendbar sind und wir deshalb auf *nichtparametrische Tests* zurückgreifen müssen. Solche Tests zur Verfügung zu haben, erweist sich als vorteilhaft, weil wir damit auch andere Statistiken vergleichen können, die nicht normalverteilt sind, beispielsweise Trainingszeiten oder die Anzahl freier Parameter.

NICHTPARAMETRISCHE TESTS

Parametrisches Tests sind im Allgemeinen robust gegenüber kleinen Abweichungen von der Normalverteilung, insbesondere, wenn die Stichprobe groß ist. Nichtparametrische Tests sind verteilungsfrei, aber weniger effizient. Wenn beide zur Verfügung stehen, sollten wir daher einen parametrischen Test vorziehen. Der entsprechende nichtparametrische Test benötigt eine größere Stichprobe, um die gleiche Leistung zu bringen. Nichtparametrische Tests setzen kein Wissen über die Verteilung der zugrundeliegenden Population voraus, sondern nur, dass die Werte verglichen oder geordnet werden können. Wie wir sehen werden, machen solche Tests von diesen Ordnungsinformationen Gebrauch.

Wenn wir einen Algorithmus haben, der auf einer Reihe von unterschiedlichen Datenmengen trainiert wurde, dann ist der Mittelwert der Fehler auf diesen Datenmengen keine Größe, der wir eine Bedeutung beimessen können, und wir können solche Mittelwerte beispielsweise auch nicht verwenden, um zwei Algorithmen A und B zu vergleichen. Das Einzige, was wir für den Vergleich zweier Algorithmen verwenden können, ist die Information, ob A auf einer der Datenmengen genauer ist als B. Wir können dann zählen, wie oft das der Fall ist, und testen, ob sich unter der

Annahme, dass sie tatsächlich gleich genau sind, der Unterschied durch den Zufall erklären lässt. Wenn wir mehr als zwei Algorithmen haben, betrachten wir die mittleren *Ränge* der mit unterschiedlichen Algorithmen trainierten Lerner. Nichtparametrische Tests verwenden grundsätzlich diese Ranginformationen und nicht die absoluten Werte.

Bevor wir mit den Details dieser Tests fortfahren, sei darauf hingewiesen, dass es nicht sinnvoll ist, Fehlerraten von Algorithmen für eine Vielzahl von Anwendungen zu vergleichen. Da es so etwa wie den „besten Lernalgorithmus“ nicht gibt, wären solche Tests nicht aufschlussreich. Allerdings können wir Algorithmen auf einer Reihe von Datenmengen (oder Versionen der gleichen Datenmenge) für dieselbe Anwendung vergleichen. Beispielsweise könnten wir mehrere Datenmengen für die Gesichtserkennung haben, die aber unterschiedliche Eigenschaften haben (etwa hinsichtlich der Auflösung, der Ausleuchtung oder der Anzahl der Subjekte), und wir wollen einen nichtparametrischen Test benutzen, um unterschiedliche Algorithmen zu vergleichen. Die unterschiedlichen Eigenschaften machen es uns unmöglich, die Bilder aus den verschiedenen Datenmengen zu einer einzigen Menge zu vereinigen, aber wir können Algorithmen separat auf den verschiedenen Datenmengen trainieren, separate Ranginformationen erhalten und diese dann kombinieren, um eine Gesamtentscheidung zu bekommen.

19.13.1 Vergleich zweier Algorithmen

Nehmen wir an, wir wollen zwei Algorithmen vergleichen. Wir trainieren beide und validieren sie auf $i = 1, \ldots, N$ verschiedenen Datenmengen, wobei wir paarweise vorgehen – d. h., alle Bedingungen außer den unterschiedlichen Algorithmen sollen identisch sein. Wir erhalten Ergebnisse e_i^1 und e_i^2 und wenn wir auf jeder Datenmenge eine K-fache-Kreuzvalidierung anwenden, sind diese die Mittelwerte oder Mediane der K-Werte. Der *Vorzeichentest* basiert auf der Idee, dass es unter der Voraussetzung, dass die beiden Algorithmen auf jeder Datenmenge den gleichen Fehler haben, eine Wahrscheinlichkeit von $1/2$ geben sollte, dass der erste Algorithmus weniger Fehler macht als der zweite. Das heißt, wir erwarten, dass der erste auf $N/2$ Datenmengen gewinnt. Wir definieren

VORZEICHENTEST

$$X_i = \begin{cases} 1 & \text{falls } e_i^1 < e_i^2 \\ 0 & \text{sonst} \end{cases} \text{ und } X = \sum_{i=1}^{N} X_i$$

und wollen die folgende Hypothese testen:

$$H_0 : \mu_1 \geq \mu_2 \text{ vs. } H_1 : \mu_1 < \mu_2$$

Falls die Nullhypothese richtig ist, ist X binomialverteilt mit N Versuchen und der Erfolgswahrscheinlichkeit $p = 1/2$. Nehmen wir an, der erste

Algorithmus gewinnt auf $X = e$ Datenmengen. Die Wahrscheinlichkeit, dass er höchstens e-mal gewinnt, wenn tatsächlich $p = 1/2$ gilt, ist

$$P(X \leq e) = \sum_{k=0}^{e} \binom{N}{x} \left(\frac{1}{2}\right)^{x} \left(\frac{1}{2}\right)^{N-x} ,$$

und wir lehnen die Nullhypothese ab, wenn diese Wahrscheinlichkeit zu klein, also kleiner als α ist. Wenn es t unentschiedene Ausgänge gibt, teilen wir diese zu gleichen Teilen auf beide Seiten auf, d. h., wir addieren $t/2$ zu e (wenn t ungerade ist, runden wir ab und verringern N um 1).

Wir testen

$$H_0 : \mu_1 \leq \mu_2 \text{ vs. } H_1 : \mu_1 > \mu_2$$

und lehnen ab, falls $P\{X \geq e\} < \alpha$.

Für den zweiseitigen Test

$$H_0 : \mu_1 = \mu_2 \text{ vs. } H_1 : \mu_1 \neq \mu_2$$

lehnen wir die Nullhypothese ab, wenn e zu klein oder zu groß ist. Im Falle $e < N/2$, lehnen wir ab, wenn $2P\{X \leq e\} < \alpha$; im Falle $e > N/2$ lehnen wir ab, wenn $2P\{X \geq e\} < \alpha$ – wir müssen die zugehörige Flanke finden und multiplizieren mit 2, weil es ein zweiseitiger Test ist.

Wie wir weiter vorn erörtert haben, können nichtparametrische Tests für den Vergleich von beliebigen Messungen verwendet werden, so zum Beispiel für Trainingszeiten. In solchen Fällen zeigt sich der Vorteil von nichtparametrischen Tests, die Ordnungsinformationen anstatt die Mittel von absoluten Werten zu verwenden. Angenommen, wie vergleichen zwei Algorithmen auf zehn Datenmengen, von denen neun klein sind und Trainingszeiten haben, die für beide Algorithmen in der Größenordnung von Minuten liegen; die zehnte Datenmenge ist sehr groß und ihre Trainingszeit liegt in der Größenordnung von einem Tag. Wenn wir einen parametrischen Test verwenden und das Mittel der Trainingszeiten bilden, dann wird die eine große Datenmenge die Entscheidung dominieren. Wenn wir dagegen mit einem nichtparametrischen Test arbeiten und für jede Datenmenge separat die Werte vergleichen, dann bewirkt die Verwendung der Ordnungsinformationen, dass für jede Datenmenge separat normiert wird, was uns dabei hilft, eine robuste Entscheidung zu treffen.

Wir können auch den Vorzeichentest als einen Stichprobentest verwenden, beispielsweise um zu prüfen, ob der mittlere Fehler auf allen Datenmengen kleiner als zwei Prozent ist, indem wir μ_1 nicht mit dem Mittelwert einer zweiten Population, sondern mit einem konstanten μ_0 vergleichen. Dazu ersetzen wir alle Beobachtungen aus einer zweiten Stichprobe durch μ_0 und führen die weiter vorn erläuterte Prozedur aus, d. h., wir zählen, wie oft wir mehr oder weniger als 0,02 bekommen und dann prüfen wir, ob das zu unwahrscheinlich unter der Nullhypothese ist. Für große N

kann die Binomialverteilung durch eine Normalverteilung approximiert werden (Übung 6), doch in der Praxis ist die Anzahl der Datenmengen eventuell kleiner als 20. Man beachte, dass der Vorzeichentest ein Test für den Median einer Population ist, welcher für symmetrische Verteilungen gleich dem Mittelwert ist.

Der Vorzeichentest verwendet nur das Vorzeichen der Differenz und nicht ihren Betrag, doch wir können uns einen Fall vorstellen, bei dem der Algorithmus, wenn er gewinnt, immer einen großen Vorsprung hat, während der zweite Algorithmus, wenn er gewinnt, immer nur ganz knapp gewinnt. Der *Wilcoxon-Vorzeichen-Rangtest* verwendet sowohl das Vorzeichen, als auch den Betrag der Differenzen.

Wilcoxon-Vorzeichen-Rangtest

Nehmen wir an, wir berechnen zusätzlich zu den Vorzeichen der Differenzen auch $m_i = |e_i^1 - e_i^2|^2$, und dann ordnen wir diese Beträge der Reihe nach, wobei der kleinste, $\min_i m_i$, den Rang 1 bekommt, der zweitkleinste den Rang 2 usw. Sollten zwei Werte gleich sein, bekommen sie den Mittelwert der beiden Ränge, die sie bekommen hätten, wenn sie sich ein wenig unterscheiden würden. Angenommen, die Beträge sind 2, 1, 2, 4, dann sind die Ränge 2,5, 1, 2,5, 4. Dann berechnen wir w_+ als die Summe aller Ränge, deren Vorzeichen positiv sind, und w_- als die Summe aller Ränge, deren Vorzeichen negativ sind.

Die Nullhypothese $\mu_1 \leq \mu_2$ kann gegenüber der Alternativhypothese $\mu_1 > \mu_2$ nur abgelehnt werden, wenn w_+ viel kleiner ist als w_-. Entsprechend kann die zweiseitige Hypothese $\mu_1 = \mu_2$ gegenüber der Alternativhypothese $\mu_1 \neq \mu_2$ nur abgelehnt werden, wenn entweder w_+ oder w_-, also $w = \min(w_+, w_-)$, sehr klein ist. Die kritischen Werte für den Wilcoxon-Vorzeichen-Rangtest sind tabelliert, und für $N > 20$ können Normalverteilungsnäherungen verwendet werden.

19.13.2 Mehrere Algorithmen

Der *Kruskal-Wallis-Test* ist die nichtparametrische Version von ANOVA und er ist eine Verallgemeinerung eines Rangtest für mehrere Stichproben. Angenommen, wir haben $M = L \cdot L$ Beoachtungen $X_{ij}, i = 1, \ldots, L, j = 1, \ldots, N$, beispielsweise Fehlerraten, aus L Algorithmen und N Datenmengen, dann ordnen wir diese Werte vom kleinsten bis zum größten und ordnen ihnen dann Ränge R_{ij} zu, die von 1 bis M gehen. Bei gleichen Werten nehmen wir wieder den Mittelwert. Wenn die Nullhypothese

Kruskal-Wallis-Test

$$H_0 : \mu_1 = \mu_2 = \cdots \mu_L$$

richtig ist, dann sollte das Mittel der Ränge von Algorithmus i näherungsweise in der Mitte zwischen 1 und M liegen, also bei $(M + 1)/2$. Wir bezeichnen das Stichprobenmittel für den Rang von Algorithmus i

mit $\bar{R}_{i\bullet}$, und wir lehnen die Hypothese ab, wenn die Stichprobenmittel für die Ränge von der Mitte abweichen. Die Teststatistik

$$H = \frac{12}{(M+1)L} \sum_{i=1}^{L} \left(\bar{R}_{i\bullet} - \frac{M+1}{2} \right)$$

ist näherungsweise Chi-Quadrat-verteilt mit $L-1$ Freiheitsgraden, und wir lehnen die Nullhypothese ab, wenn die Statistik $\mathcal{X}_{\alpha,L-1}$ übersteigt.

Ähnlich wie beim parametrischen ANOVA-Test können wir im Falle, dass die Hypothese abgelehnt wird, post-hoc-Tests für den paarweisen Vergleich der Ränge durchführen. Eine Methode hierfür ist der *Tukey-Test,* der die Rangstatistik

TUKEY-TEST

$$q = \frac{\bar{R}_{\max} - \bar{R}_{\min}}{\sigma_w}$$

verwendet. Dabei bezeichnen $\bar{R}_{\max}$ und $\bar{R}_{\min}$ den größten bzw. kleinsten Mittelwert (der Ränge) und σ_w ist die mittlere Schwankung der Ränge um das Gruppenmittel. Wir lehnen die Nullhypothese ab, dass die Gruppen i und j den gleichen Rang haben, und favorisieren die Alternativhypothese, dass sie verschieden sind, falls

$$|\bar{R}_{i\bullet} - \bar{R}_{j\bullet}| > q_\alpha(L, L(K-1))\sigma_w$$

Die $q_\alpha(L, L(K-1))$ sind tabelliert. Es können auch einseitige Tests definiert werden, um Algorithmen anhand des mittleren Rangs zu ordnen.

Demsar (2006) hat vorgeschlagen, zur Visualisierung Diagramme der kritischen Differenz zu verwenden. Auf einer Skala von 1 bis L markieren wir die Mittel $\bar{R}_{i\bullet}$ und zeichnen zwischen den Gruppen Linien, deren Länge durch die kritische Differenz $q_\alpha(L, L(K-1))$ bestimmt ist, so dass die Linien Gruppen verbinden, die statistisch nicht signifikant verschieden sind.

19.14 Multivariate Tests

Alle Tests, die wir bisher in diesem Kapitel vorgestellt haben, sind univariat, d. h., sie verwenden ein einziges Maß für die Leistungsfähigkeit, zum Beispiel den Fehler, die Präzision oder die Fläche unter der Kurve. Allerdings wissen wir, dass unterschiedliche Maße unterschiedliche Fehler bewerten. So ist etwa der Fehlklassifikationsfehler die Summe aus falsch positiven und falsch negativen Entscheidungen, und ein Test auf den Fehler kann nicht zwischen diesen unterschiedlichen Fehlerarten unterscheiden. Stattdessen kann man einen bivariater Test benutzen, der insofern mächtiger als ein univariater ist, als er auch berücksichtigt, um welchen Typ von Fehlklassifikation es sich handelt. Entsprechend können

bivariate Tests für die Kombinationen [rp-Rate, fp-Rate] oder [Präzision, Trefferquote] definiert werden, die gleichzeitig beide Maße testen (Yıldız, Aslan und Alpaydın 2011).

Angenommen, wir verwenden p Maße. Wenn wir anhand der Kombinationen [rp-Rate, fp-Rate] oder [Präzision, Trefferquote] vergleichen, haben wir $p = 2$. Tatsächlich werden alle der in Tabelle 19.2 gezeigten Leistungsmaße wie rp-Rate oder Präzision aus denselben vier Einträgen in Tabelle 19.3 berechnet, und anstatt irgendein vordefiniertes Maß zu verwenden, können wir auch einfach loslegen und einen vier-variaten Test anhand von [rp, fp, fn, rn] durchführen.

19.14.1 Vergleich zweier Algorithmen

Wir nehmen an, dass die $\boldsymbol{x}_{ij}$ p-variate Normalverteilungen sind. Wir haben $i = 1, \ldots, K$ Durchgänge und beginnen mit dem Vergleich zweier Algorithmen, so dass wir $j = 1, 2$ haben. Wir wollen testen, ob die beiden Populationen den gleichen Mittelwertvektor im p-dimensionalen Raum haben:

$$H_0 : \boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 \text{ vs. } H_1 : \boldsymbol{\mu}_1 \neq \boldsymbol{\mu}_2 \,.$$

Für das gepaarte Testen berechnen wir die Paardifferenzen $\boldsymbol{d}_i = \boldsymbol{x}_{1i} - \boldsymbol{x}_{2i}$ und testen, ob diese den Mittelwert null haben:

$$H_0 : \boldsymbol{\mu}_d = \mathbf{0} \text{ vs. } H_1 : \boldsymbol{\mu}_d \neq \mathbf{0} \,.$$

Um dies zu testen, berechnen wir das Stichprobenmittel und die Kovarianzmatrix

$$\boldsymbol{m} = \sum_{i=1}^{K} \boldsymbol{d}_i / K \,, \tag{19.26}$$

$$\mathbf{S} = \frac{1}{K-1} \sum_{i} (\boldsymbol{d}_i - \boldsymbol{m})(\boldsymbol{d}_i - \boldsymbol{m})^T \,.$$

Unter der Nullhypothese hat die *Hotelling-Statistik* für multivariate Tests

Hotelling-Statistik

$$T'^2 = K \boldsymbol{m}^T \mathbf{S}^{-1} \boldsymbol{m} \tag{19.27}$$

eine Hotelling-T^2-Verteilung mit dem Parameter p und $K - 1$ Freiheitsgraden (Rencher 1995). Wir lehnen die Nullhypothese ab, falls $T^2 > T^2_{\alpha,p,K-1}$.

Für $p = 1$ reduziert sich der Test auf den gepaarten t-Test, den wir in Abschnitt 19.11.2 diskutiert haben. In Gleichung 19.14 misst $\sqrt{K}m/S$

den normalisierten Abstand zu 0 in einer Dimension, während hier $K\boldsymbol{m}^T\mathbf{S}^{-1}\boldsymbol{m}$ den quadratischen Mahalanobis-Abstand zu $\mathbf{0}$ in p Dimensionen misst. In beiden Fällen lehnen wir ab, wenn der Abstand so groß ist, dass er höchstens in $\alpha \cdot 100$ Prozent der Fälle auftreten kann.

Wenn der multivariate Test die Nullhypothese ablehnt, können wir p separate, univariater post-hoc-Tests durchführen (unter Verwendung von Gleichung 19.14), um herauszufinden, welche Größen die Ablehnung verursachen. Wenn zum Beispiel ein multivariater Test auf [fp, fn] die Nullhypothese ablehnt, dann können wir testen, ob die Differenz auf eine signifikante Differenz bei den falsch positiven Entscheidungen, falsch negativen Entscheidungen oder beide zurückzuführen ist.

Es kann sein, dass keine der univariaten Differenzen signifikant ist, die multivariate dagegen schon – dies ist ein Vorteil multivariater Tests. Die Linearkombination der Einflussgrößen, die zur maximalen Differenz führt, ist

$$\boldsymbol{w} = \mathbf{S}^{-1}\boldsymbol{m} \,. \tag{19.28}$$

Wir können den Effekt der verschiedenen univariaten Dimensionen erkennen, wenn wir uns die entsprechenden Elemente von $\boldsymbol{w}$ anschauen. Für $p = 4$ können wir $\boldsymbol{w}$ als ein neues Leistungsmaß betrachten, das mithilfe der ursprünglichen vier Werte der Konfusionsmatrix definiert wird. Der Umstand, dass dies der Richtung in Fishers-LDA (siehe Abschnitt 6.8) entspricht, ist kein Zufall – wir suchen die Richtung, welche die Trennung der beiden Gruppen von Daten maximiert.

19.14.2 Vergleich mehrerer Algorithmen

Auf ähnliche Weise können wir einen multivariaten Test für das Vergleichen von $L > 2$ Algorithmen durch die multivariate Version von ANOVA erhalten, die MANOVA genannt wird. Wir testen

$$H_0 : \boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \cdots \boldsymbol{\mu}_L \text{ vs.}$$
$$H_1 : \boldsymbol{\mu}_r \neq \boldsymbol{\mu}_s \text{ für mindestens ein Paar } r,\, s.$$

Es sei $\boldsymbol{x}_{ij}, i = 1, \ldots, K, j = 1, \ldots, L$ der p-dimensionale Leistungsvektor von Algorithmus j auf dem Validierungsdurchgang i. MANOVA berechnet die beiden Matrizen für die Streung zwischen und innerhalb der Gruppen:

$$\mathbf{H} = K \sum_{j=1}^{L} (\boldsymbol{m}_j - \boldsymbol{m})(\boldsymbol{m}_j - \boldsymbol{m})^T \,,$$
$$\mathbf{E} = \sum_{j=1}^{L} \sum_{i=1}^{K} (\boldsymbol{x}_{ij} - \boldsymbol{m}_j)(\boldsymbol{x}_{ij} - \boldsymbol{m}_j)^T \,.$$

Die Teststatistik

$$\Lambda' = \frac{|\mathbf{E}|}{|\mathbf{E}+\mathbf{H}|} \tag{19.29}$$

hat dann eine Wilks-Λ-Verteilung mit $p, L(K-1), L-1$ (Rencher 1995). Wir lehnen die Nullhypothese ab, wenn $\Lambda' > \Lambda_{\alpha,p,L(K-1),L-1}$. Man beachte, dass die Ablehnung für kleine Werte von Λ' erfolgt: Wenn die Vektoren der Stichprobenmittel gleich sind, erwarten wir, dass $\mathbf{H}$ null ist und Λ' gegen eins geht; wenn die Stichprobenmittel weiter streuen, wird $\mathbf{H}$ „größer“ als $\mathbf{E}$ und Λ' geht gegen null.

Wenn MANOVA ablehnt, haben wir verschiedene Möglichkeiten, post-hoc-Tests durchzuführen. Eine davon ist eine Serie von paarweisen multivariaten Tests, wie wir sie zuvor diskutiert haben, mit denen wir feststellen können, welche Paare signifikant verschieden sind. Oder wir führen p separate univariate ANOVA-Tests auf den einzelnen Größen durch (siehe Abschnitt 19.12), um festzustellen, welche davon zur Ablehnung geführt haben.

Wenn MANOVA ablehnt, kann die Differenz auf eine Linearkombination der Einflussgrößen zurückzuführen sein: Die Vektoren der Mittelwerte belegen einen Raum, dessen Dimensionalität durch $S = \min(p, L-1)$ gegeben ist; seine Dimensionen sind die Eigenvektoren von $\mathbf{E}^{-1}\mathbf{H}$, und indem wir uns diese Eigenvektoren anschauen, können wir die Richtungen (neue Leistungsmaße) identifizieren, welche die Ablehnung durch MANOVA bewirken. Ist beispielsweise $\lambda_i / \sum_{i=1}^{s} \lambda_i > 0{,}9$, erhalten wir etwa eine Richtung, und das Auftragen der Projektion der Daten entlang dieser Richtung erlaubt uns eine univariate Ordnung der Algorithmen.

19.15 Anmerkungen

Der Stoff über das Design von Experimenten folgt der Diskussion in Montgomery (2005) und wurde hier für den Fall des maschinellen Lernens angepasst. Eine detailliertere Betrachtung der Intervallschätzung, der Hypothesenprüfung und der Varianzanalyse findet sich in jedem einführenden Statistikbuch, beispielsweise in Ross (1987).

Dietterich (1998) behandelt statistische Tests und vergleicht sie an einer Zahl von Anwendungen unter Nutzung verschiedener Klassifikationsalgorithmen. Eine Übersicht über den Einsatz von ROC und die Berechnung der AUC ist in Fawcett (2006) zu finden. Demsar (2006) bietet eine Zusammenstellung von statistischen Tests für das Vergleichen von Klassifikatoren über mehrere Datenmengen.

Wenn wir zwei oder mehr Algorithmen vergleichen und die Nullhypothese, dass sie die gleiche Fehlerrate haben, nicht abgelehnt wird, dann wählen wir den einfacheren Algorithmus, d. h. denjenigen, der die kleinere räumliche oder zeitliche Komplexität aufweist. Das bedeutet, dass wir unsere

a-priori-Präferenz verwenden, falls nicht die Daten einen Algorithmus anhand der Fehlerrate präferieren. Wenn wir zum Beispiel ein lineares und ein nichtlineares Modell vergleichen und der Test die Hypothese nicht ablehnt, dass sie die gleiche erwartete Fehlerrate haben, dann sollten wir uns für das einfachere Modell, also das lineare, entscheiden. Selbst im Falle einer Ablehnung durch den Test ist die Fehlerrate nur eines von mehreren Kriterien, um einen Algorithmus einem anderen vorzuziehen. Andere Kriterien wie die Trainingskomplexität (räumlich und/oder zeitlich), das Testen der Komplexität oder die Interpretierbarkeit können in der Praxis den Ausschlag geben.

Hier noch eine Anmerkung dazu, wie die Ergebnisse von post-hoc-Tests beim MultiTest (Yıldız und Alpaydın 2006) verwendet werden, um eine vollständig Ordnung zu generieren. Wir führen $L(L-1)/2$ einseitige paarweise Tests durch, um die L Algorithmen zu ordnen, aber es ist sehr wahrscheinlich, dass die Tests keine vollständige Ordnung liefern, sondern nur eine partielle. Die fehlenden Teile werden ergänzt, indem a-priori-Informationen über die Komplexität genutzt werden. Eine topologische Sortierung ergibt eine Ordnung der Algorithmen, bei der beide Arten von Information – Fehler und Komplexität – einbezogen werden.

Es gibt auch Test, die das Prüfen auf *Kontraste* erlauben. Angenommen, 1 und 2 sind Methoden, die auf neuronalen Netzen beruhen, während 3 und 4 mit Fuzzy-Logik arbeiten. Wir können dann testen, ob das Mittel aus 1 und 2 von dem Mittel aus 3 und 4 abweicht, was es uns erlaubt, die Methoden allgemein zu vergleichen.

Statistisches Vergleichen wird nicht nur benötigt, um zwischen verschiedenen Lernalgorithmen zu wählen, sondern auch, um die Parameter eines Algorithmus einzustellen. Die statistische Versuchsplanung liefert uns Werkzeuge, um dies auf effiziente Art und Weise zu tun. Beispielsweise kann die Antwortflächenmethode angewendet werden, um die Gewichte in einem Lernszenario mit mehreren Kerneln zu lernen (Gönen und Alpaydın 2011)

Ein weiterer wichtiger Punkt, den es zu beachten gilt, ist der, dass wir Fehlklassifikationen nur bewerten oder vergleichen. Dies impliziert, dass aus unserem Blickwinkel alle Fehlklassifikationen dieselben Kosten mit sich bringen. Ist dies nicht der Fall, dann sollten unsere Tests auf Risiken basieren und eine angemessene Verlustfunktion in die Betrachtungen einbeziehen. In diesem Feld wurde noch nicht viel Forschungsarbeit geleistet. Auf ähnliche Weise sollten diese Tests von der Klassifikation auf die Regression generalisiert werden, damit es möglich wird, die mittleren quadratischen Fehler der Regressionsalgorithmen zu bewerten oder damit die Fehler zweier Regressionsalgorithmen verglichen werden können.

Wenn wir zwei Klassifikationsalgorithmen vergleichen, dann müssen wir bedenken, dass wir nur testen, ob sie die gleiche erwartete Fehlerrate haben. Wenn dies der Fall ist, bedeutet das nicht, dass sie die gleichen Fehler machen. Diese Idee haben wir in Kapitel 17 ausgenutzt: Wir

können mehrere Modelle kombinieren, um die Genauigkeit zu verbessern, wenn die verschiedenen Klassifikatoren unterschiedliche Fehler machen.

19.16 Übungen

1. Gegeben ist ein Zweiklassenproblem, für dessen Verlustmatrix $\lambda_{11} = \lambda_{22} = 0, \lambda_{21} = 1$ und $\lambda_{12} = \alpha$ gilt. Bestimmen Sie den Schwellwert der Entscheidung als Funktion von α.

   LÖSUNG: Das Risiko bei der Wahl der ersten Klasse ist $0 \cdot P(C_1|x) + \alpha \cdot P(C_2|x)$ und das Risiko bei der Wahl der zweiten Klasse ist $0 \cdot P(C_1|x) + \alpha \cdot P(C_2|x)$ (Abschnitt 3.3). Wir wählen C_1, falls der erste Ausdruck kleiner ist als der zweite, und vorausgesetzt $P(C_2|x) = 1 - P(C_1|x)$, wählen wir C_1, falls

   $$P(C_1|x) > \frac{\alpha}{1+\alpha} .$$

   Das heißt, das Variieren des Schwellwerts entspricht dem Variieren der relativen Kosten von falsch positiven und falsch negativen Entscheidungen.

2. Wir können einen Klassifikator mit Fehlerwahrscheinlichkeit p simulieren, indem wir Stichproben aus einer Bernoulli-Verteilung ziehen. Tun Sie dies und implementieren Sie den Binomialtest, den approximierten Test und den t-Test für $p_0 \in (0, 1)$. Wiederholen sie diese Tests mindestens 1.000 Mal für verschiedene Werte von p und berechnen Sie die Wahrscheinlichkeit für eine Ablehnung der Nullhypothese. Welchen Wert erwarten Sie für die Wahrscheinlichkeit der Ablehnung, wenn $p_0 = p$?

3. Nehmen Sie an, dass $x^t \sim \mathcal{N}(\mu, \sigma^2)$, wobei σ^2 bekannt ist. Wie können wir $H_0 : \mu \geq \mu_0$ versus $H_1 : \mu < \mu_0$ prüfen?

   LÖSUNG: Unter H_0 haben wir

   $$z = \frac{\sqrt{N}(m - \mu_0)}{\sigma} \sim \mathcal{Z} .$$

   Wir akzeptieren H_0, falls $z \in (-z_\alpha, \infty)$.

4. Der K-fache kreuzvalidierte t-Test testet nur hinsichtlich der Gleichheit von Fehlerraten. Führt dieser Test zur Ablehnung, wissen wir nicht, welcher Klassifikationsalgorithmus die niedrigere Fehlerrate besitzt. Wie können wir prüfen, ob der erste Klassifikationsalgorithmus keine höhere Fehlerrate aufweist als der zweite? Hinweis: Wir müssen $H_0 : \mu \leq 0$ versus $H_1 : \mu > 0$ prüfen.

5. Zeigen Sie, dass die Gesamtsumme der Quadrate aufgeteilt werden kann in eine Summe von Quadraten zwischen Gruppen und eine Summe von Quadraten innerhalb der Gruppen: $SS_T = SS_b + SS_w$.

6. Verwenden Sie für den Vorzeichentest die Normalverteilungsnäherung der Binomialverteilung.

   Lösung: Unter der Nullhypothese, dass beide gleich gut sind, haben wir $p = 1/2$, und für N Datenmengen erwarten wir, dass die Anzahl X der Gewinne näherungsweise gaußverteilt ist mit $\mu = pN = N/2$ und $\sigma^2 = p(1-p)N = N/4$. Wenn es e Gewinne gibt, lehnen wir ab, falls $P(X < e) > \alpha$ oder falls $P(Z < \frac{e-N/2}{\sqrt{N/4}}) > \alpha$.

7. Nehmen wir an, uns liegen drei Klassifikationsalgorithmen vor. Wie können wir diese drei vom besten zum schlechtesten ordnen?

8. Angenommen, wir haben zwei Varianten von Algorithmus A und drei Varianten von Algorithmus B. Wie können wir die Gesamtgenauigkeiten von A und B unter Berücksichtigung all ihrer Varianten vergleichen?

   Lösung: Dazu können wir *Kontraste* verwenden (Montgomery 2005). Was wir im Wesentlichen machen, ist ein Vergleich des Mittels der beiden Varianten von A mit dem Mittel der drei Varianten von B.

9. Schlagen Sie einen geeigneten Test vor, um die Fehler der beiden Regressionsalgorithmen zu vergleichen.

   Lösung: Bei der Regression minimieren wir die Summe der Quadrate, die ein Maß für die Varianz ist, und von dieser wissen wir, dass sie eine Chi-Quadrat-Verteilung hat. Da wir den F-Test benutzen, um die Varianzen zu vergleichen (wie im Falle von ANOVA), können wir diesen auch verwenden, um die quadratischen Fehler der beiden Regressionsalgorithmen zu vergleichen.

10. Schlagen Sie einen geeigneten Test vor, um die erwarteten Belohnungen zweier Algorithmen für bestärkendes Lernen zu vergleichen.

19.17 Literaturangaben

Alpaydın, E. 1999. „Combined 5×2 cv F Test for Comparing Supervised Classification Learning Algorithms.“ *Neural Computation* 11: 1885–1892.

Bouckaert, R. R. 2003. „Choosing between Two Learning Algorithms based on Calibrated Tests.“ In *Twentieth International Conference on Machine Learning,* ed. T. Fawcett and N. Mishra, 51–58. Menlo Park, CA: AAAI Press.

Demsar, J. 2006. „Statistical Comparison of Classifiers over Multiple Data Sets.“ *Journal of Machine Learning Research* 7: 1–30.

Dietterich, T. G. 1998. „Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms.“ *Neural Computation* 10: 1895–1923.

Fawcett, T. 2006. „An Introduction to ROC Analysis.“ *Pattern Recognition Letters* 27: 861–874.

Gönen, M., and E. Alpaydın. 2011. „Regularizing Multiple Kernel Learning using Response Surface Methodology.“ *Pattern Recognition* 44: 159–171.

Montgomery, D. C. 2005. *Design and Analysis of Experiments.* 6th ed. New York: Wiley.

Rencher, A. C. 1995. *Methods of Multivariate Analysis.* New York: Wiley.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Turney, P. 2000. „Types of Cost in Inductive Concept Learning.“ Paper presented at *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning,* Stanford University, Stanford, CA, July 2.

Wolpert, D. H. 1995. „The Relationship between PAC, the Statistical Physics Framework, the Bayesian Framework, and the VC Framework.“ In *The Mathematics of Generalization,* ed. D. H. Wolpert, 117–214. Reading, MA: Addison-Wesley.

Yıldız, O. T., and E. Alpaydın. 2006. „Ordering and Finding the Best of $K > 2$ Supervised Learning Algorithms.“ *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28: 392–402.

Yıldız, O. T., Ö. Aslan, and E. Alpaydın. 2011. „Multivariate Statistical Tests for Comparing Classification Algorithms,“ In *Learning and Intelligent Optimization (LION) Conference,* ed. C. A. Coello, 1–15. Heidelberg: Springer.

Anhang

Wahrscheinlichkeit

Wir wiederholen in Kurzform die Elemente der Wahrscheinlichkeit, das Konzept einer Zufallsvariable und Beispielverteilungen.

A.1 Elemente der Wahrscheinlichkeit

Ein Zufallsexperiment ist eines, dessen Ergebnis nicht mit Gewissheit im Voraus vorhersagbar ist (Ross 1987; Casella und Berger 1990). Die Menge aller möglichen Ergebnisse wird als *Ergebnisraum* Ω bezeichnet. Ein Ergebnisraum ist *diskret*, wenn er aus einer endlichen (oder zählbar unendlichen) Menge an Ergebnissen besteht; andernfalls ist er *kontinuierlich*. Jede beliebige Teilmenge E aus Ω ist ein *Ereignis*. Ereignisse sind Mengen und wir können ihre Komplemente, Schnittmengen, Vereinigungen, usw. betrachten.

Eine Interpretation der Wahrscheinlichkeit als *relative Häufigkeit*: wenn ein Experiment kontinuierlich unter den exakt gleichen Bedingungen wiederholt wird, dann nähert sich für ein beliebiges Ereignis E das Verhältnis der Zeit, wie oft das Ereignis in E liegt, an einen konstanten Wert an. Diese konstante einschränkende relative Häufigkeit ist die Wahrscheinlichkeit des Ereignisses und wird mit $P(E)$ bezeichnet.

Manchmal wird die Wahrscheinlichkeit als *Grad des Glaubens* interpretiert. Wenn wir zum Beispiel von den Chancen der Türkei, die Fußballweltmeisterschaft 2018 zu gewinnen, sprechen, dann meinen wir damit keine Häufigkeit des Auftretens, da die Meisterschaft nur einmal stattfinden wird und das Ereignis bis jetzt (zum Zeitpunkt als dieses Buch verfasst wurde) noch nicht eingetreten ist. Was wir stattdessen in so einem Fall meinen, ist ein subjektiver Grad des Glaubens, dass das Ereignis eintreten wird. Da er subjektiv ist, können unterschiedliche Personen demselben Ereignis verschiedene Wahrscheinlichkeiten zuweisen.

A.1.1 Axiome der Wahrscheinlichkeit

Axiome stellen sicher, dass die in einem Zufallsexperiment zugewiesenen Wahrscheinlichkeiten als relative Häufigkeit interpretiert werden können und, dass die Zuweisungen mit unserem intuitiven Verständnis von Beziehungen unter relativen Häufigkeit übereinstimmen:

1. $0 \leq P(E) \leq 1$. Falls E_1 ein Ereignis ist, das auf keinen Fall eintreten kann, dann ist $P(E_1) = 0$. Falls E_2 sicher eintritt, dann ist $P(E_2) = 1$.

2. Ω ist der Ergebnisraum, der alle möglichen Ergebnisse enthält. $P(\Omega) = 1$.

3. Falls $E_i, i = 1, \ldots, n$ sich gegenseitig ausschließen (d.h., wenn sie nicht zur selben Zeit auftreten können, so wie bei $E_i \cap E_j = \emptyset, j \neq i$, wobei $\emptyset$ für das *leere Ereignis* steht, das keine möglichen Ergebnisse enthält), dann erhalten wir

$$P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{i=1}^{n} P(E_i)\,. \tag{A.1}$$

   Sei zum Beispiel E^c das *Komplement* von E, welches aus allen möglichen Ergebnissen in Ω besteht, die nicht in E liegen, dann haben wir $E \cap E^C = \emptyset$ und

$$\begin{aligned} P(E \cup E^c) &= P(E) + P(E^c) = 1 \\ P(E^c) &= 1 - P(E)\,. \end{aligned}$$

Wenn die Schnittmenge von E und F nicht leer ist, dann gilt

$$P(E \cup F) = P(E) + P(F) - P(E \cap F)\,. \tag{A.2}$$

A.1.2 Bedingte Wahrscheinlichkeit

$P(E|F)$ ist die Wahrscheinlichkeit des Eintretens von Ereignis E unter der Bedingung, dass F eingetreten ist, und sie ergibt sich als

$$P(E|F) = \frac{P(E \cap F)}{P(F)}\,. \tag{A.3}$$

Das Wissen, dass F eingetreten ist, reduziert den Ergebnisraum auf F, und dessen Teil, in dem E ebenfalls eintrat, ist $E \cap F$. Man beachte, dass Gleichung A.3 nur dann eindeutig definiert ist, wenn $P(F) > 0$. Da $\cap$ kommutativ ist, erhalten wir

$$P(E \cap F) = P(E|F)P(F) = P(F|E)P(E)\,,$$

was uns die *Bayessche Formel* liefert:

$$P(F|E) = \frac{P(E|F)P(F)}{P(E)} \,. \tag{A.4}$$

Schließen sich F_i gegenseitig aus und sind sie vollständig, also $\bigcup_{i=1}^{n} F_i = \Omega$, dann gilt

$$\begin{aligned} E &= \bigcup_{i=1}^{n} E \cap F_i \,, \\ P(E) &= \sum_{i=1}^{n} P(E \cap F_i) = \sum_{i=1}^{n} P(E|F_i)P(F_i) \,. \end{aligned} \tag{A.5}$$

Die Bayessche Formel erlaubt uns die Notation

$$P(F_i|E) = \frac{P(E \cap F_i)}{P(E)} = \frac{P(E|F_i)P(F_i)}{\sum_j P(E|F_j)P(F_j)} \,. \tag{A.6}$$

Sind E und F *unabhängig*, dann erhalten wir $P(E|F) = P(E)$ und somit

$$P(E \cap F) = P(E)P(F) \,. \tag{A.7}$$

Das heißt, die Kenntnis dessen, dass F eingetreten ist, verändert nicht die Wahrscheinlichkeit, dass E eintritt.

A.2 Zufallsvariablen

Eine *Zufallsvariable* ist eine Funktion, die jedem Ergebnis – enthalten im Ergebnisraum – eines Zufallsexperiments einen Wert zuweist.

A.2.1 Funktionen der Wahrscheinlichkeitsverteilung und Wahrscheinlichkeitsdichte

Die *Funktion $F(\cdot)$ einer Wahrscheinlichkeitsverteilung* einer Zufallsvariable X für eine beliebige reelle Zahl a ist

$$F(a) = P\{X \leq a\} \tag{A.8}$$

und wir erhalten

$$P\{a < X \leq b\} = F(b) - F(a) \,. \tag{A.9}$$

Ist X eine diskrete Zufallsvariable, so gilt

$$F(a) = \sum_{\forall x \leq a} P(x) \,, \tag{A.10}$$

wobei $P(\cdot)$ die *Wahrscheinlichkeitsmassefunktion* ist, die durch $P(a) = P\{X = a\}$ definiert wird. Ist X eine *kontinuierliche* Zufallsvariable, dann ist $p(\cdot)$ die *Wahrscheinlichkeitsdichtefunktion*, so dass

$$F(a) = \int_{-\infty}^{a} p(x)dx \,. \tag{A.11}$$

A.2.2 Gemeinsame Verteilungs- und Dichtefunktionen

In bestimmten Experimenten interessiert uns möglicherweise die Beziehung zwischen zwei oder mehreren Zufallsvariablen, und wir nutzen die *gemeinsamen* Wahrscheinlichkeitsverteilungs- und Dichtefunktionen von X und Y unter Beachtung, dass

$$F(x, y) = P\{X \leq x, Y \leq y\} \,. \tag{A.12}$$

Individuelle *Randverteilungen* und Randdichten können durch Marginalisierung berechnet werden, also durch die Summierung über die freie Variable:

$$F_X(x) = P\{X \leq x\} = P\{X \leq x, Y \leq \infty\} = F(x, \infty) \,. \tag{A.13}$$

Im diskreten Fall schreiben wir

$$P(X = x) = \sum_j P(x, y_j) \tag{A.14}$$

und im kontinuierlichen Fall erhalten wir

$$p_X(x) = \int_{-\infty}^{\infty} p(x, y)dy \,. \tag{A.15}$$

Sind X und Y *unabhängig*, dann gilt

$$p(x, y) = p_X(x)p_Y(y) \,. \tag{A.16}$$

Diese können analog auf mehr als zwei Zufallsvariablen generalisiert werden.

A.2.3 Bedingte Verteilungen

Sind X und Y Zufallsvariablen, dann ist

$$\begin{aligned} P_{X|Y}(x|y) &= P\{X = x|Y = y\} \\ &= \frac{P\{X = x, Y = y\}}{P\{Y = y\}} = \frac{P(x, y)}{P_Y(y)} \,. \end{aligned} \tag{A.17}$$

A.2.4 Satz von Bayes

Wenn zwei Zufallsvariablen gemeinsam verteilt sind, wobei der Wert von einer bekannt ist, dann ist die Wahrscheinlichkeit, dass die andere einen festgelegten Wert annimmt mit Hilfe des *Satzes von Bayes* berechenbar:

$$P(y|x) = \frac{P(x|y)P_Y(y)}{P_X(x)} = \frac{P(x|y)P_Y(y)}{\sum_y P(x|y)P_Y(y)} \,. \tag{A.18}$$

Oder in Worten ausgedrückt:

$$\begin{aligned} &\text{a-posteriori-Wahrscheinlichkeit} \\ &\quad = \frac{\text{Likelihood} \times \text{a-priori-Wahrscheinlichkeit}}{\text{Evidenz}} . \end{aligned} \tag{A.19}$$

Man beachte, dass der Nenner durch Summieren (oder Integrieren, falls y kontinuierlich ist) des Zählers über alle möglichen Werte von y gewonnen wird. Die „Form“ von $p(y|x)$ hängt vom Zähler ab, wobei der Nenner als Normalisierungsfaktor agiert, um zu garantieren, dass $p(y|x)$ sich zu 1 summiert. Der Satz von Bayes erlaubt uns, eine a-priori-Wahrscheinlichkeit in eine a-posteriori-Wahrscheinlichkeit zu überführen, indem durch x bereitgestellte Informationen einbezogen werden.

Der Satz von Bayes kehrt die Abhängigkeiten um, wodurch es uns ermöglicht wird, $p(y|x)$ zu berechnen, wenn $p(x|y)$ bekannt ist. Angenommen, dass y die „Ursache“ von x ist; zum Beispiel sei y ein Sommerurlaub und x stehe für einen Sonnenbrand. Dann ist $p(x|y)$ die Wahrscheinlichkeit, dass jemand, von dem bekannt ist, dass er im Sommerurlaub war, einen Sonnenbrand hat. Dies ist der *kausale* (oder prädiktive) Weg. Der Satz von Bayes erlaubt uns einen *diagnostischen* Ansatz, indem es uns möglich ist, $p(y|x)$ zu berechnen, sprich, die Wahrscheinlichkeit, dass jemand, von dem bekannt ist, dass er einen Sonnenbrand hat, im Sommerurlaub war. Dann ist $p(y)$ die allgemeine Wahrscheinlichkeit, dass irgendjemand im Sommerurlaub war und $p(x)$ ist die Wahrscheinlichkeit, dass irgendjemand einen Sonnenbrand hat, inklusive derer, die im Sommerurlaub waren und derer, die das nicht waren.

A.2.5 Erwartung

Die *Erwartung*, der *Erwartungswert* oder der *Mittelwert* einer Zufallsvariable X, bezeichnet mit $E[X]$, ist der Durchschnittswert von X bei einer großen Anzahl an Experimenten:

$$E[X] = \begin{cases} \sum_i x_i P(x_i) & \text{falls } X \text{ diskret ist ,} \\ \int x p(x) dx & \text{falls } X \text{ kontinuierlich ist .} \end{cases} \tag{A.20}$$

Er ist ein gewichteter Durchschnitt, bei dem jeder Wert durch die Wahrscheinlichkeit gewichtet wird, dass X diesen Wert annimmt. Er hat die folgenden Eigenschaften ($a, b \in \mathbb{R}$):

$$\begin{aligned} E[aX + b] &= aE[X] + b \\ E[X + Y] &= E[X] + E[Y] . \end{aligned} \tag{A.21}$$

Für jede beliebige Funktion $g(\cdot)$ mit reellem Wert ist der Erwartungswert gleich

$$E[g(X)] = \begin{cases} \sum_i g(x_i) P(x_i) & \text{falls } X \text{ diskret ist ,} \\ \int g(x) p(x) dx & \text{falls } X \text{ kontinuierlich ist .} \end{cases} \tag{A.22}$$

Der Sonderfall $g(x) = x^n$, bezeichnet als das n-te Moment von X, definiert sich als

$$E[X^n] = \begin{cases} \sum_i x_i^n P(x_i) & \text{falls } X \text{ diskret ist ,} \\ \int x^n p(x) dx & \text{falls } X \text{ kontinuierlich ist .} \end{cases} \tag{A.23}$$

Der *Mittelwert* ist das erste Moment und wird mit μ bezeichnet.

A.2.6 Varianz

Die *Varianz* misst, wie stark X um den Erwartungswert herum variiert. Wenn $\mu \equiv E[X]$, dann definiert sich die Varianz als

$$\text{Var}(X) = E[(X - \mu)^2] = E[X^2] - \mu^2 . \tag{A.24}$$

Die Varianz ist das zweite Moment minus das Quadrat des ersten Moments. Die Varianz, bezeichnet mit σ^2, erfüllt die folgende Eigenschaft ($a, b \in \mathbb{R}$):

$$\text{Var}(aX + b) = a^2 \text{Var}(X) . \tag{A.25}$$

Mit $\sqrt{\text{Var}(X)}$ bezeichnet man die *Standardabweichung*, die durch σ dargestellt wird. Die Standardabweichung hat die gleiche Einheit wie X und ist leichter zu interpretieren als die Varianz.

Die *Kovarianz* deutet die Beziehung zwischen zwei Zufallsvariablen an. Wenn das Eintreten von X die Wahrscheinlichkeit des Eintretens von Y vergrößert, dann ist die Kovarianz positiv; sie ist negativ, wenn das Eintreten von X die Wahrscheinlichkeit des Eintretens von Y verringert, und sie ist gleich 0, wenn keinerlei Abhängigkeit existiert.

$$\mathrm{Cov}(X,Y) = E\left[(X-\mu_X)(Y-\mu_Y)\right] = E[XY] - \mu_X\mu_Y \qquad \text{(A.26)}$$

mit $\mu_X \equiv E[X]$ und $\mu_Y \equiv E[Y]$. Einige andere Eigenschaften sind

$$\begin{aligned}
\mathrm{Cov}(X,Y) &= \mathrm{Cov}(Y,X) \\
\mathrm{Cov}(X,X) &= \mathrm{Var}(X) \\
\mathrm{Cov}(X+Z,Y) &= \mathrm{Cov}(X,Y) + \mathrm{Cov}(Z,Y) \\
\mathrm{Cov}\left(\sum_i X_i, Y\right) &= \sum_i \mathrm{Cov}(X_i,Y) \qquad \text{(A.27)}
\end{aligned}$$

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X,Y) \qquad \text{(A.28)}$$

$$\mathrm{Var}\left(\sum_i X_i\right) = \sum_i \mathrm{Var}(X_i) + \sum_i \sum_{j\neq i} \mathrm{Cov}(X_i,X_j)\,. \qquad \text{(A.29)}$$

Sind X und Y unabhängig, dann ist $E[XY] = E[X]E[Y] = \mu_X\mu_Y$ und $\mathrm{Cov}(X,Y) = 0$. Somit gilt, wenn X_i unabhängig sind:

$$\mathrm{Var}\left(\sum_i X_i\right) = \sum_i \mathrm{Var}(X_i)\,. \qquad \text{(A.30)}$$

Die *Korrelation* ist eine normalisierte, dimensionslose Quantität, die immer zwischen -1 und $+1$ liegt:

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}\,. \qquad \text{(A.31)}$$

A.2.7 Das schwache Gesetz großer Zahlen

Sei $\mathcal{X} = \{X^t\}_{t=1}^N$ eine Menge an unabhängigen und identisch verteilten Zufallsvariablen, von denen jede den Mittelwert μ und eine finite Varianz σ^2 aufweist. Dann gilt für ein beliebiges $\epsilon > 0$

$$P\left\{\left|\frac{\sum_t X^t}{N} - \mu\right| > \epsilon\right\} \to 0 \text{ as } N \to \infty\,. \qquad \text{(A.32)}$$

Das heißt, der Durchschnitt von N Versuchen konvergiert zum Mittelwert, wenn sich N erhöht.

A.3 Spezielle Zufallsvariablen

Es existieren gewisse Arten von Zufallsvariablen, die so häufig auftreten, dass ihnen Namen gegeben wurden.

A.3.1 Bernoulli-Verteilung

Ein Versuch wird durchgeführt, dessen Ergebnis entweder „Erfolg“ oder „Misserfolg“ ist. Die Zufallsvariable X ist eine 0/1-Indikatorvariable und nimmt den Wert 1 für ein erfolgreiches Ergebnis an und in den anderen Fällen den Wert 0. p ist die Wahrscheinlichkeit, dass das Ergebnis des Versuchs ein Erfolg ist. Dann gilt

$$P\{X = 1\} = p \text{ und } P\{X = 0\} = 1 - p\,, \tag{A.33}$$

was äquivalent auch notiert werden kann als

$$P\{X = i\} = p^i(1-p)^{1-i}, i = 0, 1\,. \tag{A.34}$$

Ist X eine Bernoulli-verteilte Zufallsvariable, dann ergibt sich ihr Erwartungswert und ihre Varianz als:

$$E[X] = p,\ \mathrm{Var}(X) = p(1-p)\,. \tag{A.35}$$

A.3.2 Binomialverteilung

Werden N identische unabhängige Bernoulli-Versuche durchgeführt, dann ist die Zufallsvariable X, welche die Anzahl an Erfolgen, die in N Versuchen eintreten, repräsentiert, binomialverteilt. Die Wahrscheinlichkeit, dass es i Erfolge gibt, ist

$$P\{X = i\} = \binom{N}{i} p^i(1-p)^{N-i}, i = 0 \ldots N\,. \tag{A.36}$$

Ist X binomialverteilt, dann ergibt sich ihr Erwartungswert und ihre Varianz als

$$E[X] = Np,\ \mathrm{Var}(X) = Np(1-p)\,. \tag{A.37}$$

A.3.3 Multinomiale Verteilung

Betrachten wir eine Verallgemeinerung der Bernoulli-Verteilung. Das Ergebnis eines Zufallsexperiments ist nicht mehr einer von zwei Zuständen, sondern nun ist das Ergebnis einer von K sich gegenseitig ausschließenden und vollständigen Zuständen, von denen jeder die Wahrscheinlichkeit des Eintretens von p_i mit $\sum_{i=1}^{K} p_i = 1$ besitzt. Angenommen, dass N

derartige Versuche durchgeführt werden, bei denen das Ergebnis i genau N_i Mal mit $\sum_{i=1}^{k} N_i = N$ eintritt. Dann ist die gemeinsame Verteilung von $N_1, N_2, \ldots, N_K$ multinomial:

$$P(N_1, N_2, \ldots, N_K) = N! \prod_{i=1}^{K} \frac{p_i^{N_i}}{N_i!} \,. \tag{A.38}$$

Ein Sonderfall liegt vor, wenn $N = 1$; nur ein Versuch wird durchgeführt. Dann sind N_i hier 0/1-Indikatorvariablen, von denen nur eine gleich 1 ist und alle anderen gleich 0 sind. Dann reduziert sich Gleichung A.38 auf

$$P(N_1, N_2, \ldots, N_K) = \prod_{i=1}^{K} p_i^{N_i} \,. \tag{A.39}$$

A.3.4 Gleichverteilung

X ist gleichverteilt über das Intervall $[a, b]$, wenn seine Dichtefunktion durch

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{falls } a \leq x \leq b \,, \\ 0 & \text{andernfalls} \end{cases} \tag{A.40}$$

gegeben ist.

Ist X gleichverteilt, dann sind ihr Erwartungswert und ihre Varianz gleich

$$E[X] = \frac{a+b}{2}, \ \mathrm{Var}(X) = \frac{(b-a)^2}{12} \,. \tag{A.41}$$

A.3.5 Normalverteilung (Gauß-Verteilung)

X ist normalverteilt oder Gauß-verteilt mit Mittelwert μ und Varianz σ^2, bezeichnet mit $\mathcal{N}(\mu, \sigma^2)$, wenn sich ihre Dichtefunktion darstellen läSSt als:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], -\infty < x < \infty \,. \tag{A.42}$$

Viele Zufallsphänomene folgen der glockenförmigen Normalverteilung zumindest annähernd, und viele Beobachtungen aus der Natur können als kontinuierliche, leicht verschiedene Versionen eines typischen Werts angesehen werden – deshalb spricht man hier von der *Normalverteilung*. In so einem Fall repräsentiert μ den typischen Wert und σ definiert, wie stark die Instanzen um den prototypischen Wert herum variieren.

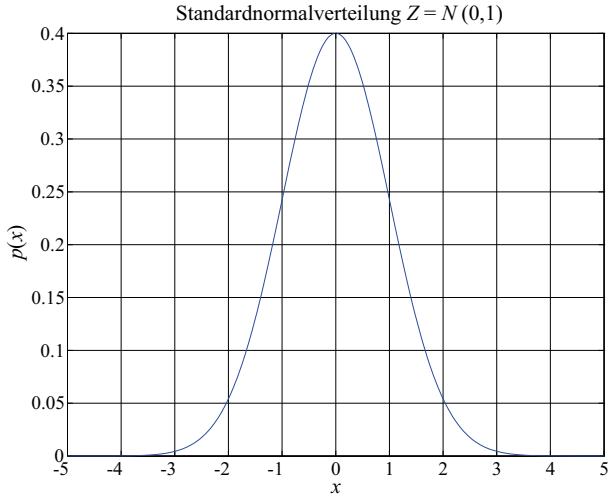


***Abb. A.1*:** Wahrscheinlichkeitsdichtefunktion von $\mathcal{Z}$, der Standardnormalverteilung.

68,27% liegen in $(\mu - \sigma, \mu + \sigma)$, 95,45% in $(\mu - 2\sigma, \mu + 2\sigma)$ und 99,73% fallen in $(\mu - 3\sigma, \mu + 3\sigma)$. Somit ist $P\{|x - \mu| < 3\sigma\} \approx 0{,}99$. Für praktische Zwecke ist $p(x) \approx 0$, falls $x < \mu - 3\sigma$ oder $x > \mu + 3\sigma$. $\mathcal{Z}$ ist standardnormalverteilt, sprich, $\mathcal{N}(0, 1)$ (siehe Abbildung A.1) und die zugehörige Dichte ergibt sich als

$$p_Z(x) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{x^2}{2}\right] . \tag{A.43}$$

Falls $X \sim \mathcal{N}(\mu, \sigma^2)$ und $Y = aX + b$, dann $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$. Die Summe von unabhängigen normalverteilten Variablen ist ebenfalls normalverteilt mit $\mu = \sum_i \mu_i$ und $\sigma^2 = \sum_i \sigma_i^2$. Falls X gleich $\mathcal{N}(\mu, \sigma^2)$, dann gilt

$$\frac{X - \mu}{\sigma} \sim \mathcal{Z} . \tag{A.44}$$

Hierbei spricht man von der z-Normalisierung.

ZENTRALER GRENZWERTSATZ

Seien $X_1, X_2, \ldots, X_N$ eine Menge an unabhängigen und identisch verteilten Zufallsvariablen, die alle den Mittelwert μ und die Varianz σ^2 besitzen. Dann legt der *zentrale Grenzwertsatz* fest, dass für große N die Verteilung von

$$X_1 + X_2 + \ldots + X_N \tag{A.45}$$

annähernd $\mathcal{N}(N\mu, N\sigma^2)$ ist. Ist X zum Beispiel binomial mit den Parametern (N, p), dann kann X als die Summe von N Bernoulli-Versuchen

geschrieben werden und $(X - Np)/\sqrt{Np(1-p)}$ ist annähernd standardnormalverteilt.

Der zentrale Grenzwertsatz wird auch verwendet, um normalverteilte Zufallsvariablen auf Computern zu generieren. Programmiersprachen besitzen Subroutinen, die einheitlich verteilte (pseudo-)zufällige Zahlen im Bereich $[0, 1]$ liefern. Wenn U_i solche Zufallsvariablen sind, dann ist $\sum_{i=1}^{12} U_i - 6$ annähernd $\mathcal{Z}$.

Sei $X^t \sim \mathcal{N}(\mu, \sigma^2)$. Der geschätzte Stichprobenmittelwert

$$m = \frac{\sum_{t=1}^{N} X^t}{N} \tag{A.46}$$

ist ebenfalls normalverteilt mit Mittelwert μ und Varianz σ^2/N.

A.3.6 Chi-Quadrat-Verteilung

Sind Z_i unabhängige standardnormalverteilte Zufallsvariablen, dann ist

$$X = Z_1^2 + Z_2^2 + \ldots + Z_n^2 \tag{A.47}$$

eine Chi-Quadrat-Verteilung mit n Freiheitsgraden, sprich $X \sim \mathcal{X}_n^2$, mit

$$E[X] = n, \ \mathrm{Var}(X) = 2n \, . \tag{A.48}$$

Wenn $X^t \sim \mathcal{N}(\mu, \sigma^2)$, dann ist die geschätzte Stichprobenvarianz gleich

$$S^2 = \frac{\sum_t (X^t - m)^2}{N - 1} \tag{A.49}$$

und wir erhalten

$$(N-1)\frac{S^2}{\sigma^2} \sim \mathcal{X}_{N-1}^2 \, . \tag{A.50}$$

Es ist außerdem bekannt, dass m und S^2 unabhängig sind.

A.3.7 t-Verteilung

Falls $Z \sim \mathcal{Z}$ und $X \sim \mathcal{X}_n^2$ unabhängig sind, dann ist

$$T_n = \frac{Z}{\sqrt{X/n}} \tag{A.51}$$

t-verteilt mit n Freiheitsgraden mit

$$E[T_n] = 0, n > 1, \ \mathrm{Var}(T_n) = \frac{n}{n-2}, n > 2 \, . \tag{A.52}$$

Wie die Standardnormalverteilung ist t symmetrisch um 0 herum. Mit wachsendem n ähnelt die t-Dichte mehr und mehr der Standardnormalverteilung mit dem Unterschied, dass t dickere Schweife hat, welche auf stärkere Variabilität als bei der Normalverteilung hinweisen.

A.3.8 F-Verteilung

Wenn $X_1 \sim \mathcal{X}^2_n$ und $X_2 \sim \mathcal{X}^2_m$ unabhängige Chi-Quadrat-Zufallsvariablen mit n bzw. m Freiheitsgraden sind, dann ist

$$F_{n,m} = \frac{X_1/n}{X_2/m} \tag{A.53}$$

F-verteilt mit n und m Freiheitsgraden mit

$$E[F_{n,m}] = \frac{m}{m-2}, m > 2, \ \mathrm{Var}(F_{n,m}) = \frac{m^2(2m+2n-4)}{n(m-2)^2(m-4)}, m > 4\,. \tag{A.54}$$

A.4 Literaturangaben

Casella, G., and R. L. Berger. 1990. *Statistical Inference.* Belmont, CA: Duxburry.

Ross, S. M. 1987. *Introduction to Probability and Statistics for Engineers and Scientists.* New York: Wiley.

Index