e-bol.net

Lassen Sie User-Daten für sich arbeiten

Algorithmen für smartere Web 2.0-Apps

Kollektive Intelligenz

analysieren, programmieren & nutzen

Toby Segaran

Deutsche Übersetzung von Thomas Demmig

O'REILLY®

e-bol.net

e-bol.net

Kollektive Intelligenz analysieren, programmieren und nutzen

Toby Segaran

Deutsche Übersetzung von Thomas Demmig

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

e-bol.net

Die Informationen in diesem Buch wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag, Autoren und Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene Fehler und deren Folgen.
Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Der Verlag richtet sich im Wesentlichen nach den Schreibweisen der Hersteller. Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Alle Rechte vorbehalten einschließlich der Vervielfältigung, Übersetzung, Mikroverfilmung sowie Einspeicherung und Verarbeitung in elektronischen Systemen.

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

Copyright der deutschen Ausgabe:
© 2008 by O'Reilly Verlag GmbH & Co. KG
1. Auflage 2008

Die Originalausgabe erschien 2007 unter dem Titel
Programming Collective Intelligence bei O'Reilly Media, Inc.

Die Darstellung von Königspinguinen im Zusammenhang mit dem
Thema Kollektive Intelligenz ist ein Warenzeichen von O'Reilly Media, Inc.

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über *http://dnb.ddb.de* abrufbar.

Übersetzung und deutsche Bearbeitung: Thomas Demmig, Mannheim
Lektorat: Alexandra Follenius, Köln
Fachgutachten: Prof. Dr. Jörg Schütz, Homburg
Korrektorat: Sibylle Feldmann, Düsseldorf
Satz: III-satz, Husby
Umschlaggestaltung: Karen Montgomery, Sebastopol & Michael Oreal, Köln
Produktion: Andrea Miß, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; www.koeselbuch.de

ISBN-13 978-3-89721-780-5

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

e-bol.net

Inhalt

Einleitung IX

1 Einführung in die kollektive Intelligenz 1
Was ist kollektive Intelligenz? 2
Was ist maschinelles Lernen? 4
Grenzen des maschinellen Lernens 5
Beispiele aus dem richtigen Leben 5
Andere Anwendungen für lernende Algorithmen 6

2 Empfehlungen geben 9
Kollaboratives Filtern 10
Vorlieben sammeln 10
Ähnliche Benutzer finden 12
Dinge empfehlen 18
Produkte finden 20
Link-Empfehlungen mit del.icio.us erstellen 22
Elementbasiertes Filtern 26
Verwenden der MovieLens-Daten 29
Benutzerbasiertes oder elementbasiertes Filtern? 31
Übungen 32

3 Gruppen bilden 33
Überwachtes versus unüberwachtes Lernen 34
Wortvektoren 34
Hierarchische Clusteranalyse 37
Zeichnen des Dendrogramms 43
Spalten-Clusteranalyse 45
K-Means-Clusteranalyse 47
Cluster mit Vorlieben 50

e-bol.net

Daten in zwei Dimensionen betrachten 55
Andere Dinge in der Clusteranalyse 58
Übungen 59

4 Suchen und Bewerten 61
Was gehört zu einer Such-Engine? 61
Ein einfacher Crawler 63
Aufbauen des Index 66
Abfragen 71
Contentbasierte Bewertung 72
Eingehende Links verwenden 78
Aus Klicks lernen 83
Übungen 94

5 Optimierung 96
Gruppenreisen 97
Lösungen repräsentieren 98
Die Zielfunktion 99
Random Search 102
Bergsteigen 103
Simulierte Abkühlung 106
Genetische Algorithmen 108
Echte Flugsuchen 112
Optimierung nach Vorlieben 118
Netzwerkvisualisierung 122
Andere Möglichkeiten 128
Übungen 128

6 Dokumente filtern 130
Spam filtern 131
Dokumente und Wörter 131
Trainieren des Klassifizierers 133
Wahrscheinlichkeiten berechnen 135
Ein naiver Klassifizierer 137
Die Fisher-Methode 142
Die trainierten Klassifizierer persistieren 146
Blog-Feeds filtern 148
Merkmalserkennung verbessern 151
Akismet 153
Alternative Methoden 155
Übungen 156

e-bol.net

7 Modellieren mit Entscheidungsbäumen . . . 158
Anmeldungen vorhersagen . . . 158
Entscheidungsbäume . . . 161
Trainieren des Baums . . . 162
Die beste Aufteilung wählen . . . 163
Rekursiver Aufbau des Baums . . . 166
Anzeigen des Baums . . . 168
Klassifikation neuer Beobachtungen . . . 171
Zurechtstutzen des Baums . . . 173
Umgang mit fehlenden Daten . . . 175
Umgang mit numerischen Ergebnissen . . . 176
Modellieren von Immobilienpreisen . . . 177
Modellieren der »Hotness« . . . 180
Wann nutzt man Entscheidungsbäume? . . . 183
Übungen . . . 184

8 Aufbauen von Preismodellen . . . 186
Aufbau eine Beispiel-Datenmenge . . . 187
k-nächste Nachbarn . . . 188
Gewichtete Nachbarn . . . 192
Kreuzvalidierung . . . 196
Heterogene Variablen . . . 198
Optimieren der Skalierung . . . 202
Ungleiche Verteilungen . . . 203
Nutzung echter Daten – die eBay-API . . . 210
Wann man k-nächste Nachbarn nutzt . . . 217
Übungen . . . 217

9 Komplexe Klassifikation: Kernel-Methoden und SVMs . . . 219
Datenmenge für Vermittler . . . 219
Schwierigkeiten mit den Daten . . . 221
Einfache lineare Klassifikation . . . 224
Kategoriale Eigenschaften . . . 228
Skalieren der Daten . . . 233
Kernel-Methoden verstehen . . . 234
Support-Vektor-Maschinen . . . 238
Verwenden der LIBSVM . . . 240
Vermitteln bei Facebook . . . 243
Übungen . . . 249

e-bol.net

10 Unabhängige Merkmale finden **251**
Eine Nachrichtensammlung 252
Vorangegangene Ansätze 256
Nicht-negative Matrix-Faktorisierung 259
Anzeigen der Ergebnisse 266
Daten des Aktienmarkts nutzen 270
Übungen 276

11 Entwickeln von Intelligenz **277**
Was ist genetische Programmierung? 278
Programme als Bäume 280
Erzeugen der initialen Population 285
Testen einer Lösung 286
Mutieren von Programmen 288
Crossover 291
Aufbau der Umgebung 293
Ein einfaches Spiel 296
Weitere Möglichkeiten 301
Übungen 304

12 Zusammenfassung der Algorithmen **306**
Bayes-Klassifizierer 306
Entscheidungsbaum-Klassifizierer 310
Neuronale Netze 315
Support-Vektor-Maschinen 319
k-nächste Nachbarn 324
Clusteranalyse 328
Multidimensionales Skalieren 332
Nicht-negative Matrix-Faktorisierung 334
Optimierung 337

A Fremdbibliotheken **340**

B Mathematische Formeln **347**

Index **355**

e-bol.net

Einleitung

Die wachsende Zahl derjenigen, die sich im Internet tummeln, sei es regelmäßig oder nur gelegentlich, hat zum Entstehen einer großen Datenmenge geführt, die uns Millionen verschiedener Einblicke in Benutzererfahrung, Marketing, persönliche Vorlieben und menschliches Verhalten im Allgemeinen ermöglicht. Dieses Buch bietet eine Einführung in den wachsenden Bereich der *kollektiven Intelligenz*. Es stellt Wege vor, interessante Daten von Websites zu holen, von denen Sie vielleicht gehört haben, enthält Ideen dazu, wie Daten von Benutzern Ihrer eigenen Anwendungen zu sammeln sind, und zeigt viele verschiedene Arten auf, die Daten zu analysieren und zu verstehen, nachdem Sie sie gefunden haben.

Das Ziel dieses Buchs ist, Sie über die einfachen datenbankbasierten Anwendungen hinauszuführen und zu erklären, wie Sie schlauere Programme schreiben, um von den Informationen, die Sie und andere täglich sammeln, zu profitieren.

Voraussetzungen

Die Codebeispiele in diesem Buch sind in Python geschrieben, und ein paar Grundkenntnisse in der Python-Programmierung könnten hilfreich sein, aber ich erkläre alle Algorithmen so, dass auch Programmierer anderer Sprachen sie verstehen können. Der Python-Code wird insbesondere von denjenigen, die High-Level-Sprachen wie Ruby oder Perl kennen, leicht zu lesen sein. Dieses Buch soll keine Anleitung dazu sein, wie man Programmieren lernt, daher ist es wichtig, dass Sie schon genug Programme geschrieben haben, um mit den grundlegenden Konzepten vertraut zu sein. Wenn Sie Rekursion verstehen und einfache funktionale Programmierung kennen, wird es für Sie noch einfacher sein.

Dieses Buch geht nicht davon aus, dass Sie sich mit Datenanalysen, maschinellem Lernen oder Statistik auskennen. Ich habe versucht, mathematische Konzepte so einfach wie möglich zu erklären, aber grundlegende Kenntnisse der Trigonometrie und ein bisschen Statistikwissen werden Ihnen dabei helfen, die Algorithmen besser zu verstehen.

e-bol.net

Form der Beispiele

Die Codebeispiele in den Kapiteln sind in Lehrbuchform geschrieben, wodurch Sie angeregt werden sollen, die Anwendungen schrittweise aufzubauen, und lernen, wie die Algorithmen funktionieren. In den meisten Fällen werden Sie eine neue Funktion oder Methode nach ihrem Erstellen in einer interaktiven Session nutzen, um zu verstehen, wie sie funktioniert. Die Algorithmen sind meistens einfache Varianten, die in vielerlei Hinsicht erweitert werden können. Indem Sie die Beispiele durcharbeiten und interaktiv testen, werden Sie Einblicke in die Möglichkeiten zur Verbesserung Ihrer eigenen Anwendungen bekommen.

Warum Python?

Auch wenn die Algorithmen im Text beschrieben und die Formeln erläutert werden, ist es sinnvoller (und vermutlich auch verständlicher), richtigen Code für die Algorithmen und Beispielprobleme zu haben. Alle Beispiele in diesem Buch sind in Python geschrieben, einer ausgezeichneten High-Level-Sprache. Ich habe Python gewählt, weil es viele positive Eigenschaften besitzt:

Kürzer
: Code, der in dynamisch typisierten Sprachen wie Python geschrieben wird, ist meist kürzer als Code, der in anderen verbreiteten Sprachen entsteht. Das bedeutet, dass Sie weniger tippen müssen, wenn Sie sich durch die Beispiele arbeiten, es heißt aber auch, dass es einfacher ist, den Algorithmus in Ihrem Kopf unterzubringen und wirklich zu verstehen, was er tut.

Leicht zu lesen
: Python wurde schon als »ausführbarer Pseudocode« beschrieben. Das ist natürlich übertrieben, aber es stimmt sehr wohl, dass die meisten erfahrenen Programmierer Python-Code lesen und verstehen können, was er tun soll. Manche der weniger eingängigen Konstrukte in Python werden im Abschnitt »Python-Tipps« beschrieben.

Einfach erweiterbar
: Python bringt standardmäßig vielen Bibliotheken mit, unter anderem mit mathematischen Funktionen, zum Parsen von XML (Extensible Markup Language) und zum Laden von Webseiten. Die in diesem Buch verwendeten Bibliotheken, die nicht zum Standard gehören, wie zum Beispiel der RSS-(Really Simple Syndication-)Parser und die SQLite-Schnittstelle, sind frei erhältlich und lassen sich einfach herunterladen, installieren und nutzen.

Interaktiv
: Wenn Sie ein Beispiel durcharbeiten, ist es hilfreich, die Funktionen schon auszuprobieren, während Sie sie schreiben, ohne ein anderes Programm nur zum

e-bol.net

Testen erstellen zu müssen. Python kann Programme direkt von der Befehlszeile aus ausführen und besitzt zudem einen interaktiven Prompt, an dem Sie Funktionen starten, Objekte erstellen und Pakete testen können.

Viele Paradigmen
: Python unterstützt objektorientierte, prozedurale und funktionale Programmierung. Algorithmen zum maschinellen Lernen sind sehr unterschiedlich, und der geradlinigste Weg bei der Implementierung des einen Algorithmus kann sich sehr von dem eines anderen unterscheiden. Manchmal ist es sinnvoll, Funktionen als Parameter zu übergeben, während man ein anderes Mal den Zustand in einem Objekt ablegt. Python unterstützt beide Vorgehensweisen.

Viele Plattformen und freie Software
: Python bietet eine Referenzimplementierung für alle wichtigen Plattformen und ist auf allen frei erhältlich. Der Code, der in diesem Buch beschrieben wird, funktioniert unter Windows, Linux und auf dem Macintosh.

Python-Tipps

Einsteigern, die etwas über die Programmierung in Python lernen wollen, empfehle ich das Buch *Einführung in Python* von Mark Lutz und David Ascher (O'Reilly), das einen ausgezeichneten Überblick bietet. Programmierer anderer Sprachen sollten den Python-Code recht leicht lesen können. Allerdings werde ich in diesem Buch auch speziellere Python-Syntax verwenden, da ich damit Algorithmen oder grundlegende Konzepte besser ausdrücken kann. Hier ein kurzer Überblick für diejenigen unter Ihnen, die keine Python-Programmierer sind:

Listen- und Verzeichniskonstruktoren

Python bietet eine gute Auswahl einfacher Typen, wobei zwei besonders häufig in diesem Buch verwendet werden: *list* und *dictionary*. Eine List ist eine geordnete Liste mit beliebigen Werten und wird mit eckigen Klammern konstruiert:

```
number_list=[1,2,3,4]
string_list=['a', 'b', 'c', 'd']
mixed_list=['a', 3, 'c', 8]
```

Ein Dictionary ist eine ungeordnete Menge von Schlüssel-Wert-Paaren, ähnlich einer Hash-Map in anderen Sprachen. Es wird mit geschweiften Klammern definiert:

```
ages={'John':24,'Sarah':28,'Mike':31}
```

Die Elemente von Lists und Dictionaries können über eckige Klammern hinter dem Listennamen angesprochen werden:

```
string_list[2]   # gibt 'b' zurück
ages['Sarah']    # gibt 28 zurück
```

e-bol.net

Leerzeichen sind wichtig

Anders als die meisten Sprachen verwendet Python Einrückungen im Code, um Codeblöcke zu definieren. Schauen Sie sich diesen Codeabschnitt an:

```
if x==1:
  print 'x ist 1'
  print 'Immer noch im if-Block'
print 'Außerhalb des if-Blocks'
```

Der Interpreter weiß, dass die ersten beiden `print`-Anweisungen ausgeführt werden, wenn `x` den Wert `1` hat, weil der Code eingerückt ist. Einrückungen können beliebig viele Leerzeichen enthalten, solange die Zahl konsistent ist. Dieses Buch verwendet beim Einrücken zwei Leerzeichen. Wenn Sie Code eingeben, müssen Sie darauf achtgeben, dass die Einrückungen korrekt kopiert werden.

List Comprehensions

Eine *List Comprehension* ist eine praktische Möglichkeit, eine Liste in eine andere umzuwandeln und dabei Filter und Funktionen auf die Elemente anzuwenden. Eine List Comprehension wird geschrieben als:

```
[Ausdruck for Variable in Liste]
```

oder:

```
[Ausdruck for Variable in Liste if Bedingung]
```

So würde der folgende Code:

```
l1=[1,2,3,4,5,6,7,8,9]
print [v*10 for v in l1 if v1>4]
```

zum Beispiel diese Liste ausgeben:

```
[50,60,70,80,90]
```

List Comprehensions werden in diesem Buch oft genutzt, weil es sich dabei um einen außerordentlich direkten Weg handelt, eine Funktion auf eine ganze Liste anzuwenden oder unerwünschte Elemente auszusortieren. Außerdem werden sie häufig im Konstruktor von `dict` verwendet:

```
l1=[1,2,3,4,5,6,7,8,9]
timesten=dict([(v,v*10) for v in l1])
```

Dieser Code erzeugt ein Dictionary mit der ursprünglichen Liste als Schlüssel und jedem Element multipliziert mit 10 als Wert:

```
{1:10,2:20,3:30,4:40,5:50,6:60,7:70,8:80,9:90}
```

e-bol.net

Offene APIs

Die Algorithmen, um kollektive Intelligenz zu synthetisieren, erfordern Daten von vielen Benutzern. Neben den Algorithmen zum maschinellen Lernen behandelt dieses Buch eine Reihe von Open Web-APIs (Application Programming Interfaces). Dabei handelt es sich um Wege, die Firmen Ihnen anbieten, um frei auf die Daten ihrer Websites über ein bestimmtes Protokoll zuzugreifen – Sie können dann Programme schreiben, um die Daten herunterzuladen und zu verarbeiten. Diese Daten enstehen meistens aus Beiträgen, die die Anwender der Site eingebracht haben und die nun nach neuen Einsichten durchforstet werden können. In manchen Fällen gibt es eine Python-Bibliothek, um auf diese APIs zuzugreifen – wenn nicht, ist es recht einfach, Ihre eigene Schnittstelle zu erstellen, um auf die Daten mithilfe der bei Python mitgelieferten Bibliothek zum Herunterladen von Daten und zum Parsen von XML zuzugreifen.

Hier ein paar der Websites mit offenen APIs, die Sie in diesem Buch vorfinden werden:

del.icio.us
: Eine Social Bookmarking Application, über deren offene API Sie Verweise nach Tags oder von einem bestimmten Benutzer herunterladen können.

Kayak
: Eine Reise-Site mit einer API, um Flug- und Hotelsuchen in Ihren eigenen Programmen durchzuführen.

eBay
: Eine Site für Onlineauktionen, die es Ihnen erlaubt, nach gerade zum Verkauf stehenden Objekten zu suchen.

Hot or Not
: Eine Rating- und Dating-Site mit einer API, um nach Menschen zu suchen und deren Ratings und demografische Informationen zu erhalten.

Akismet
: Eine API für kollaboratives Spam-Filtern.

Sie können eine große Zahl potenzieller Anwendungen bauen, indem Sie Daten einer einzelnen Quelle verarbeiten, sie aus mehreren Quellen zusammenbringen und sogar indem Sie externe Informationen mit Beiträgen Ihrer eigenen Benutzer kombinieren. Die Möglichkeit, Daten zu nutzen, die auf verschiedenste Art und auf unterschiedlichen Sites von den Leuten eingegeben wurden, ist ein zentraler Bestandteil beim Erstellen kollektiver Intelligenz. Ein guter Ausgangspunkt beim Suchen nach weiteren Websites mit offenen APIs ist ProgrammableWeb (*http://www.programmableweb.com*).

e-bol.net

Überblick über die Kapitel

Jeder Algorithmus in diesem Buch hat ein realistisches Problem zur Grundlage, das (hoffentlich) von allen Lesern leicht verstanden wird. Ich habe versucht, Probleme zu vermeiden, bei denen man viel über das Umfeld wissen muss, und mich auf Probleme konzentriert, die – auch wenn sie sich als komplex darstellen – den meisten Leuten vertraut sind.

Kapitel 1, *Einführung in die kollektive Intelligenz*

Erläutert die Konzepte des maschinellen Lernens, wie es in den verschiedenen Bereichen angewandt wird und wie es genutzt werden kann, um neue Schlussfolgerungen aus Daten zu ziehen, die von vielen verschiedenen Leuten zusammengetragen wurden.

Kapitel 2, *Empfehlungen geben*

Stellt die *kollaborativen Filtertechniken* vor, die von vielen Onlinehändlern genutzt werden, um Produkte oder Medien zu empfehlen. Das Kapitel enthält einen Abschnitt über Link-Empfehlungen auf einer Social Bookmarking-Site und das Erstellen eines Film-Empfehlungssystems aus den MovieLens-Daten.

Kapitel 3, *Gruppen bilden*

Baut auf einigen der Ideen aus Kapitel 2 auf und stellt zwei verschiedene Methoden des *Clusterings* vor, die automatisch Gruppen gleicher Objekte in einer großen Datenmenge finden. Dieses Kapitel zeigt die Verwendung des Clusterns zum Finden von Gruppen in einer Menge beliebter Weblogs und in den Vorlieben von Nutzern einer Social Networking-Website.

Kapitel 4, *Suchen und Bewerten*

Beschreibt die verschiedenen Teile einer Such-Engine, einschließlich Crawler, Indexer und Query-Engine. Es behandelt den *PageRank*-Algorithmus zum Bewerten von Seiten in Abhängigkeit von Links, die auf sie verweisen, und zeigt Ihnen, wie Sie ein *neuronales Netz* erstellen, das lernt, welche Schlüsselwörter mit verschiedenen Ergebnissen verbunden sind.

Kapitel 5, *Optimierung*

Stellt Algorithmen zur *Optimierung* vor, die dazu entworfen sind, Millionen von möglichen Lösungen eines Problems zu durchsuchen und die beste zu wählen. Der große Anwendungsbereich dieser Algorithmen wird durch Beispiele belegt, die zum Inhalt haben, die besten Flüge für eine Gruppe von Personen mit dem gleichen Ziel zu finden, die beste Verteilung von Studenten auf die Zimmer eines Wohnheims zu ermitteln und ein Netzwerk zu erstellen, das möglichst wenige Kreuzungspunkte enthält.

Kapitel 6, *Dokumente filtern*

Führt *Bayes'sche Filter* vor, die in vielen freien und kommerziellen Spam-Filtern zum automatischen Klassifizieren von Dokumenten in Abhängigkeit von den Wörtern und anderen Eigenschaften der Dokumente verwendet werden. Das

e-bol.net

wird auf eine Menge von RSS-Suchergebnissen angewendet, um die automatische Klassifizierung der Einträge vorzunehmen.

Kapitel 7, *Modellieren mit Entscheidungsbäumen*
Stellt *Entscheidungsbäume* als Methode vor, um Voraussagen zu treffen und um den Weg zu modellieren, auf dem eine Entscheidung erfolgt. Der erste Entscheidungsbaum wird aus hypothetischen Serverlog-Daten erstellt und verwendet, um vorherzusagen, ob ein Benutzer ein Premium-Subscriber wird oder nicht. Die anderen Beispiele verwenden Daten aus realen Websites, um Immobilienpreise und die »Hotness« zu modellieren.

Kapitel 8, *Aufbauen von Preismodellen*
Geht das Problem an, keine Klassifizierungen, sondern numerische Werte mithilfe von *k-nächste Nachbarn*-Techniken zu ermitteln, und wendet die Optimierungsalgorithmen aus Kapitel 5 an. Diese Methoden werden zusammen mit der eBay-API genutzt, um ein System zu erstellen, das mögliche Auktionspreise für Gegenstände – basierend auf einer Reihe von Eigenschaften – ermittelt.

Kapitel 9, *Komplexe Klassifikation: Kernel-Methoden und SVMs*
Zeigt, wie *Support-Vektor-Maschinen* genutzt werden können, um Menschen bei Onlinedating-Sites oder bei der Suche nach beruflichen Kontakten zusammenzubringen. Support-*Vektor-Maschinen* bieten recht anspruchsvollen Stoff, und dieses Kapitel vergleicht sie mit anderen Methoden.

Kapitel 10, *Unabhängige Merkmale finden*
Führt eine recht neue Technik namens *Nicht-negative Matrix-Faktorisierung* ein, die genutzt wird, um unabhängige Eigenschaften einer Datenmenge zu ermitteln. In vielen Datensätzen entstehen die Daten als Kombination verschiedener Merkmale, die wir im Voraus nicht kennen – die Idee ist hier, diese Merkmale zu finden. Diese Technik wird auf eine Sammlung von Nachrichten angewandt, wobei die Artikel selbst verwendet werden, um die Themen zu bestimmen, von denen eines oder mehrere eventuell zu einer gegebenen Story passen.

Kapitel 11, *Entwickeln von Intelligenz*
Stellt *genetische Programmierung* vor, eine sehr anspruchsvolle Reihe von Techniken, die über die Optimierung hinausgehen und Algorithmen mithilfe von evolutionären Ideen entwickeln, um ein bestimmtes Problem zu lösen. Das wird durch ein einfaches Spiel demonstriert, in dem der Computer zunächst ein sehr schlechter Spieler ist, seine Fähigkeiten aber mit jeder weiteren Runde verbessert, indem er seinen Code anpasst.

Kapitel 12, *Zusammenfassung der Algorithmen*
Fasst alle Algorithmen zum maschinellen Lernen und zur Statistik zusammen, die in diesem Buch beschrieben werden, und vergleicht sie mit einer Reihe von theoretischen Problemen. Das wird Ihnen dabei helfen zu verstehen, wie sie funktionieren und wie sie Daten unterteilen.

e-bol.net

Anhang A, *Fremdbibliotheken*
: Gibt Ihnen Informationen über die Bibliotheken, die in diesem Buch verwendet werden – wo Sie sie erhalten und wie man sie installiert.

Anhang B, *Mathematische Formeln*
: Enthält Formeln, Beschreibungen und Codeabschnitte für viele der mathematischen Konzepte, die im Buch vorgestellt wurden.

Übungen am Ende jedes Kapitels sollen Sie auf Wege hinweisen, wie Sie die Algorithmen erweitern und für Ihre Zwecke anpassen können.

Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Normaler Text
: Menütitel, Menüoptionen, Buttons und Tasten (wie Alt und Strg).

Kursiv
: Neue Begriffe, URLs, E-Mail-Adressen, Dateinamen, Dateierweiterungen, Pfadnamen, Verzeichnisse und Unix-Tools.

`Nichtproportionalschrift`
: Befehle, Optionen, Schalter, Variablen, Attribute, Schlüssel, Funktionen, Typen, Klassen, Namensräume, Methoden, Module, Eigenschaften, Parameter, Werte, Objekte, Ereignisse, Event-Handler, XML-Tags, HTML-Tags, Makros, Dateiinhalte oder die Ausgabe von Befehlen.

`Nichtproportionalschrift fett`
: Befehle oder anderer Text, der so vom Benutzer eingegeben werden sollte.

`Nichtproportionalschrift kursiv`
: Text, der vom Benutzer durch eigene Werte ersetzt werden sollte.

Dieses Icon weist auf einen Tipp, einen Vorschlag oder eine allgemeine Anmerkung hin.

Die Codebeispiele zu diesem Buch

Es gibt eine deutsche und eine amerikanische Website von O'Reilly, auf denen Sie Beispiele, Errata und weitere Informationen zu diesem Buch finden können:

http://www.oreilly.de/catalog/collintellger/
http://www.oreilly.com/catalog/9780596529321/

e-bol.net

Die in diesem Buch abgedruckten Codebeispiele stehen dort in einer englischsprachigen Version zum Herunterladen zur Verfügung. Bitte klicken Sie auf den Beispiel-Link auf der Katalogseite zum Buch.

Die Codebeispiele verwenden

Dieses Buch soll Ihnen helfen, Ihre Arbeit zu tun. Im Allgemeinen können Sie den Code aus diesem Buch in Ihren Programmen und Dokumentationen verwenden. Solange Sie nicht gerade einen wesentlichen Anteil des Codes wiedergeben, müssen Sie uns nicht um Erlaubnis bitten. Beispielsweise erfordert das Schreiben eines Programms, in das einige Codestücke aus diesem Buch integriert werden, keine Erlaubnis. Für das Verkaufen oder Verteilen einer CD-ROM mit Beispielen aus O'Reilly-Büchern *brauchen* Sie dagegen eine Erlaubnis. Das Beantworten einer Frage, indem dieses Buch zitiert und Beispielcode angegeben wird, erfordert keine Erlaubnis. Das Einbauen einer wesentlichen Menge von Beispielcode in die Dokumentation Ihres Produkts *erfordert* dagegen eine Erlaubnis.

Wir begrüßen Quellennennung, bestehen aber nicht darauf. Eine Nennung umfasst normalerweise den Titel, den Autor, den Verlag und die ISBN. Zum Beispiel »*Kollektive Intelligenz analysieren, programmieren und nutzen* von Toby Segaran. Copyright 2008 O'Reilly-Verlag, ISBN 978-3-89721-780-5«.

Falls Sie das Gefühl haben, dass sich Ihre Verwendung von Codebeispielen außerhalb der hier erteilten Erlaubnis bewegt, dann schreiben Sie an *kommentar@oreilly.de*.

Danksagung

Ich möchte jedem bei O'Reilly meinen Dank aussprechen, der an der Entwicklung und Produktion dieses Buchs beteiligt war. Zuallererst möchte ich Nat Torkington dafür danken, dass er mir klargemacht hat, dass die Idee sinnvoll wäre und es sich lohnte, sie weiterzuverfolgen. Dann Mike Hendrickson und Brian Jepson dafür, dass sie mir zugehört und mich dazu ermuntert haben, das Buch zu schreiben. Mary O'Brien, die das Buch als Lektorin von Brian übernahm und immer wieder meine Ängste zerstreuen konnte, dass das Projekt mir über den Kopf wachsen würde.

Aus dem Produktionsteam möchte ich Marlowe Shaeffer, Rob Romano, Jessamyn Read, Amy Thomson und Sarah Schneider dafür danken, dass sie meine Zeichnungen und Texte in etwas verwandelt haben, was man tatsächlich auch anschauen möchte.

Vielen Dank an jeden, der bei der Korrektur des Buchs beteiligt war, insbesondere Paul Tyma, Matthew Russell, Jeff Hammerbacher, Terry Camerlengo, Andreas Weigend, Daniel Russell und Tim Wolters.

Vielen Dank an meine Eltern.

e-bol.net

Schließlich schulde ich vielen meiner Freunde Dank, die mir dabei geholfen haben, Ideen für das Buch zu entwickeln, und die immer Verständnis hatten, wenn ich keine Zeit für sie hatte: Andrea Matthews, Jeff Beene, Laura Miyakawa, Neil Stroup und Brooke Blumenstein. Es wäre viel schwerer gewesen, dieses Buch zu schreiben, wenn ich eure Unterstützung nicht gehabt hätte – und manche der hübschen Beispiele würden fehlen.

e-bol.net

KAPITEL 1

Einführung in die kollektive Intelligenz

Netflix ist ein Online-DVD-Verleih, über den man sich Filme nach Hause schicken lassen kann. Zudem gibt Netflix basierend auf den bisher ausgeliehenen Filmen Empfehlungen. Ende 2006 lobte die Firma einen Preis von einer Million Dollar für die erste Person aus, die die Genauigkeit des Empfehlungssystems um zehn Prozent steigern würde, dazu regelmäßige Preise von $50.000 für den in einem Jahr jeweils Führenden – bis der Wettbewerb vorüber ist. Tausende Teams aus der ganzen Welt machten sich an diese Aufgabe heran, und im April 2007 hatte die beste Mannschaft eine Verbesserung von sieben Prozent erreicht. Durch Berücksichtigung der Filme, die jeder Kunde gut fand, kann Netflix anderen Kunden Filme empfehlen, von denen diese noch nie gehört haben, und dafür sorgen, dass die Kunden weitere Filme ausleihen. Jede Verbesserung ihres Empfehlungssystems ist Netflix eine Menge Geld wert.

Die Such-Engine Google startete im Jahr 1998. Zu dieser Zeit gab es schon viele große Such-Engines, und viele gingen davon aus, dass ein neuer Konkurrent niemals an diese Giganten herankommen würde. Die Gründer von Google verfolgten allerdings einen komplett neuen Ansatz beim Bewerten von Suchergebnissen, indem sie die Links von Millionen Websites nutzten, um zu entscheiden, welche Seiten am relevantesten waren. Die Suchergebnisse von Google waren so viel besser als die der anderen Such-Engines, dass im Jahr 2004 schon 85% aller Suchen im Web über Google ausgeführt wurden. Seine Gründer gehören mittlerweile zu den zehn reichsten Leuten auf Welt.

Was haben diese beiden Firmen gemeinsam? Sie zogen beide neue Schlussfolgerungen und schafften dadurch neue geschäftliche Möglichkeiten, indem sie anspruchsvolle Algorithmen nutzten, um Daten zu kombinieren, die von vielen verschiedenen Menschen kamen. Die Fähigkeit, Informationen zu sammeln, und die Computerleistung, diese zu interpretieren, wurden perfekt kombiniert und sorgten für ein besseres Verständnis von Benutzern und Kunden. Diese Art von Arbeit wird an vielen Stellen durchgeführt – Dating-Sites wollen dabei helfen, schneller einen passen-

e-bol.net

den Partner zu finden, es entstehen Firmen, die die Änderung von Ticketpreisen für Flüge vorhersagen, und eigentlich will jeder seine Kunden besser verstehen, um Werbung gezielter einsetzen zu können.

Es gibt noch ein paar mehr Beispiele im spannenden Feld der kollektiven Intelligenz, und das Entstehen neuer Services bedeutet, dass es überall neue Möglichkeiten gibt. Ich glaube, dass das Verständnis maschinellen Lernens und statistischer Methoden in immer mehr Bereichen immer wichtiger wird, aber besonders aber beim Interpretieren und Organisieren der riesigen Informationsmengen, die von Personen auf der ganzen Welt erstellt werden.

Was ist kollektive Intelligenz?

Der Begriff *kollektive Intelligenz* wird schon seit Jahrzehnten genutzt und wurde mit dem Aufkommen neuer Kommunikationstechnologien zunehmend bekannter und wichtiger. Auch wenn der Begriff möglicherweise Gedanken über gemeinsames Bewusstsein oder übernatürliche Phänomene nahelegt, geht es im Zusammenhang mit Technologie jedoch um die Kombination aus Verhalten, Vorlieben und Ideen einer Gruppe von Leuten, um neue Einblicke zu erhalten.

Kollektive Intelligenz war natürlich auch schon möglich, bevor es das Internet gab. Man braucht das Web nicht, um Daten verschiedener Personengruppen zu sammeln, zu kombinieren und zu analysieren. Eine der einfachsten Formen davon ist eine Umfrage oder Zählung. Das Sammeln von Antworten großer Personengruppen ermöglicht es, statistische Schlussfolgerungen über die Gruppe zu ziehen, die kein einzelnes Mitglied der Gruppe selbst hätte herausfinden können. Das Finden neuer Erkenntnisse über unabhängige Teilnehmer ist genau das, worum es bei kollektiver Intelligenz geht.

Ein sehr bekanntes Beispiel bieten die Finanzmärkte, bei denen ein Preis nicht von einer einzelnen Person oder durch koordinierte Aktivitäten festgelegt wird, sondern durch das Handelsverhalten vieler unabhängiger Personen, die alle so handeln, wie es ihrer Meinung nach für sie am besten ist. Auch wenn es sich eigentlich widersinnig anhört, scheinen *Futures Markets*, in denen viele Teilnehmer Verträge auf Grundlage ihrer Einschätzungen zukünftiger Preise aushandeln, beim Vorhersagen von Preisen besser zu sein als Experten, die unabhängig davon Projektionen erstellen. Das liegt daran, dass bei diesen Märkten das Wissen, die Erfahrung und die Einblicke Tausender von Menschen kombiniert werden, um eine Vorhersage zu treffen, anstatt sich auf die Ansichten einer einzelnen Person zu verlassen.

Auch wenn Methoden zur kollektiven Intelligenz schon vor dem Internet existierten, hat die Möglichkeit, Informationen von Tausenden oder Millionen von Menschen im Web zu sammeln, viele neue Wege erschlossen. Die Menschen haben das Internet schon immer genutzt, um Käufe zu tätigen, Nachforschungen anzustellen,

e-bol.net

sich unterhalten zu lassen und ihre eigenen Websites zu bauen. Alle diese Verhaltensweisen können beobachtet werden und dann dazu dienen, Informationen davon abzuleiten, ohne die Wege der Benutzer zu stören, indem man ihnen Fragen stellt. Es gibt sehr viele Möglichkeiten, diese Informationen zu verarbeiten und zu interpretieren. Hier zwei Beispiele, die die teilweise sehr gegensätzlichen Ansätze demonstrieren:

- *Wikipedia* ist eine Online-Enzyklopädie, die komplett durch die Beiträge von Anwendern entstanden ist. Jede Seite kann von jedermann angelegt oder bearbeitet werden, und es gibt eine kleine Gruppe von Administratoren, die auf wiederholten Missbrauch achten. Wikipedia hat mehr Einträge als jede andere Enzyklopädie, und trotz gelegentlich vorhandener Manipulationen durch böswillige Zeitgenossen gehen die meisten doch davon aus, dass sie in vielen Bereichen sehr genau ist. Das ist ein Beispiel für kollektive Intelligenz, weil jeder Artikel von einer großen Gruppe von Menschen verwaltet wird und das Ergebnis ein Lexikon ist, das größer sein kann, als es eine koordinierte Gruppe jemals geschafft hätte. Die Wikipedia-Software stellt mit den Beiträgen nichts irgendwie Intelligentes an – es protokolliert nur die Änderungen und zeigt die letzte Version an.
- *Google*, das weiter oben schon erwähnt wurde, ist die weltweit beliebteste Such-Engine für das Internet, und es war die erste Such-Engine, die Webseiten aufgrund der Anzahl der Seiten bewertet hat, die auf sie verweisen. Diese Bewertungsmethode nutzt Informationen darüber, was Tausende von Menschen über eine bestimmte Webseite geschrieben haben, und legt damit einen Rang bei den Suchergebnissen fest. Das ist ein gänzlich anderes Beispiel kollektiver Intelligenz. Während Wikipedia die Leute explizit zur Teilnahme einlädt, extrahiert Google die wichtigen Informationen aus dem, was die Ersteller von Webinhalten auf ihren eigenen Sites gemacht haben, um daraus Bewertungen für ihre eigenen Benutzer zu erstellen.

Obwohl Wikipedia eine großartige Informationsquelle und ein beeindruckendes Beispiel kollektiver Intelligenz ist, ist es doch sehr viel mehr von den Benutzern abhängig, die diese Informationen beitragen, als von cleveren Algorithmen in der Software. Dieses Buch konzentriert sich auf das andere Ende des Spektrums und behandelt Algorithmen wie den PageRank von Google, der Benutzerdaten erhält und damit Berechnungen durchführt, um neue Informationen zu erzeugen, die den Anwendern helfen können. Manche Daten werden explizit eingesammelt, zum Beispiel indem Anwender gebeten werden, Dinge zu bewerten, während andere Daten nebenbei entstehen, zum Beispiel durch die Beobachtung von Kaufverhalten. In beiden Fällen ist es nicht nur wichtig, die Informationen zu sammeln und anzuzeigen, sondern auch sie intelligent zu verarbeiten und neue Informationen zu generieren.

Dieses Buch wird Ihnen Wege aufzeigen, wie Sie Daten über offene APIs sammeln können, und es wird eine ganze Reihe von Algorithmen zum maschinellen Lernen

e-bol.net

und zu statistischen Methoden vorstellen. Diese Kombination wird es Ihnen ermöglichen, Methoden zur kollektiven Intelligenz zu erstellen, die auf Daten zurückgreifen, die Sie in Ihren eigenen Anwendungen gesammelt haben. Zudem werden Sie Daten aus anderen Quellen sammeln und damit experimentieren können.

Was ist maschinelles Lernen?

Maschinelles Lernen ist ein Teilbereich der künstlichen Intelligenz (KI, im Englischen »Artificial Intelligence«, AI) der sich mit Algorithmen befasst, die es Computern erlauben zu *lernen*. Das bedeutet in den meisten Fällen, dass einem Algorithmus Daten gegeben werden, aus denen er Informationen über die Eigenschaften der Daten ermittelt – womit man Voraussagen über andere Daten treffen kann, die eventuell erst in Zukunft entstehen. Das ist deshalb möglich, weil so gut wie alle nicht zufälligen Daten Muster enthalten, aufgrund deren der Rechner verallgemeinern kann. Um zu verallgemeinern, wird ein *Modell* mit den Aspekten der Daten trainiert, die als wichtig ermittelt wurden.

Um zu verstehen, wie die Modelle ins Spiel kommen, stellen Sie sich ein einfaches Beispiel im ansonsten komplexen Feld des E-Mail-Filterns vor. Angenommen, Sie erhalten eine Menge Spam, der die Wörter »online pharmacy« enthält. Als Mensch haben Sie die nötigen Fähigkeiten, um Muster schnell zu erkennen. Sie stellen also bald fest, dass jede Nachricht mit den Wörtern »online pharmacy« Spam ist und direkt in den Mülleimer befördert werden sollte. Das ist eine Verallgemeinerung – Sie haben tatsächlich ein geistiges Modell dazu erstellt, was Spam ist. Nachdem Sie mehrere dieser Nachrichten als Spam gekennzeichnet haben, sollte ein Algorithmus aus dem Bereich des maschinellen Lernens dazu in der Lage sein, die gleiche Verallgemeinerung zu treffen.

Es gibt viele verschiedene Algorithmen zum maschinellen Lernen, alle mit unterschiedlichen Stärken und dazu gedacht, verschiedene Problembereiche zu behandeln. Manche, wie zum Beispiel Entscheidungsbäume, sind transparent, sodass ein Beobachter den Schlussfolgerungsprozess vollständig nachvollziehen kann, den der Rechner vornimmt. Andere, wie zum Beispiel neuronale Netze, sind *Black Boxes*, das heißt, es gibt eine Antwort, aber es ist oft schwer nachzuvollziehen, warum die Entscheidung so gefallen ist.

Viele Algorithmen zum maschinellen Lernen basieren stark auf Mathematik und Statistik. Nach der Definition, die ich weiter oben gegeben habe, kann man sogar sagen, dass sowohl einfache Korrelationsanalysen als auch Regression grundlegende Formen des maschinellen Lernens sind. Dieses Buch geht nicht davon aus, dass der Leser viel über Statistik weiß, daher habe ich versucht, die verwendeten statistischen Methoden so einfach wie möglich zu beschreiben.

e-bol.net

Grenzen des maschinellen Lernens

Maschinelles Lernen hat auch seine Schwächen. Die Algorithmen sind unterschiedlich gut dazu in der Lage, über große Mengen an Mustern zu verallgemeinern, und ein Muster, das zuvor noch nicht vom Algorithmus erkannt wurde, wird leicht fehlinterpretiert. Menschen haben eine Menge kulturelles Wissen und Erfahrung, um Schlussfolgerungen zu ziehen, und sind erstaunlich gut darin, ähnliche Situationen zu erkennen, wenn es darum geht, bei neuen Informationen Entscheidungen zu treffen, aber Rechner können nur aufgrund der Daten verallgemeinern, die sie schon kennen, und selbst dabei nur sehr eingeschränkt vorgehen.

Die Spam-Filter-Methode, die Sie in diesem Buch kennenlernen werden, basiert auf dem Vorhandensein von Wörtern oder Phrasen – unabhängig davon, was sie bedeuten oder in welchem Zusammenhang sie stehen. Auch wenn es theoretisch möglich ist, einen Algorithmus zu bauen, der auch die Grammatik in Betracht zieht, geschieht dies in der Praxis sehr selten, da der nötige Aufwand im Vergleich zur Verbesserung des Algorithmus bei Weitem zu groß wäre. Ein Verständnis für die Bedeutung von Wörtern oder ihre Relevanz für eine Person würde weitaus mehr Informationen erfordern, als Spam-Filter in ihrer momentanen Form bekommen könnten.

Zudem haben alle Methoden des maschinellen Lernens einen Hang zum Über-Verallgemeinern, wenn auch mit unterschiedlicher Ausprägung. Wie bei den meisten Dingen im Leben sind starke Verallgemeinerungen, die nur auf ein paar Beispielen basieren, sehr selten exakt. Es ist sicherlich möglich, dass Sie eine wichtige E-Mail von einem Freund erhalten, die die Wörter »online pharmacy« enthält. In diesem Fall würden Sie dem Algorithmus mitteilen, dass die Nachricht kein Spam ist, wodurch der Algorithmus vielleicht folgern würde, dass Nachrichten von dieser bestimmten Person akzeptabel seien. In der Natur vieler Algorithmen beim maschinellen Lernen liegt es, dass sie weiterlernen können, sobald neue Informationen eintreffen.

Beispiele aus dem richtigen Leben

Es gibt etliche Sites im Internet, die Daten von vielen verschiedenen Menschen sammeln und Methoden des maschinellen Lernens und der Statistik nutzen, um daraus Vorteile zu ziehen. Google ist dabei vermutlich eine der größten – es verwendet nicht nur Weblinks, um Seiten zu bewerten, sondern sammelt fortlaufend Informationen darüber, wann Werbung von den verschiedenen Benutzern angeklickt wird, wodurch es in die Lage versetzt wird, diese Werbung effektiver einzusetzen. In Kapitel 4 werden Sie mehr über Such-Engines und den PageRank-Algorithmus erfahren – ein wichtiges Element des Bewertungssystems von Google.

e-bol.net

Andere Beispiele sind Websites mit Empfehlungssystemen. Sites wie Amazon und Netflix nutzen Informationen darüber, was die Leute kaufen oder mieten, um zu bestimmen, welche Personen oder Objekte einander gleichen. Dadurch lassen sich dann Empfehlungen aufgrund der Kaufhistorie aussprechen. Andere Sites, wie Pandora oder Last.fm, nutzen Ihre Bewertungen zu verschiedenen Bands und Liedern, um eigene Radiostationen mit einer Musikauswahl aufzusetzen, die Ihnen hoffentlich gefällt. Kapitel 2 behandelt die Möglichkeiten, Empfehlungssysteme zu erstellen.

Prognosemärkte sind ebenfalls eine Form der kollektiven Intelligenz. Einer der bekanntesten Märkte ist Hollywood Stock Exchange (*http://hsx.com*), bei dem die Leute mit »Aktien« zu Filmen und Schauspielern handeln. Sie können eine Aktie zum aktuellen Preis kaufen oder verkaufen, wobei der abschließende Wert einem Millionstel der tatsächlichen Einnahmen des Films an den Kinokassen zum Eröffnungswochenende (oder einer gewissen Zeit darüber hinaus) entspricht. Der Preis entsteht durch das Handeln vieler Personen und nicht nur durch eine einzelne – der aktuelle Preis kann also als gemeinsame Voraussage der gesamten Gruppe zu den Einnahmen des Films gesehen werden. Die Voraussagen durch Hollywood Stock Exchange sind regelmäßig besser als die von individuellen Experten.

Manche Dating-Sites, wie zum Beispiel eHarmony, verwenden von den Teilnehmern zusammengesammelte Informationen, um zu ermitteln, wer gut zu wem passen würde. Auch wenn diese Firmen dazu tendieren, ihre Methoden zum Zusammenbringen von Personen geheim zu halten, ist es sehr wahrscheinlich, dass zu jedem erfolgreichen Ansatz eine erneute Evaluierung gehört – abhängig davon, ob das gewählte Paar wirklich zusammenpasste oder nicht.

Andere Anwendungen für lernende Algorithmen

Die in diesem Buch beschriebenen Methoden sind nicht neu, und auch wenn sich die Beispiele auf internetbasierte Probleme kollektiver Intelligenz beziehen, kann Wissen über Algorithmen zum maschinellen Lernen für Softwareentwickler in vielen anderen Bereichen ebenfalls hilfreich sein – insbesondere dort, wo mit großen Datenmengen umgegangen wird, die nach interessanten Mustern durchsucht werden können. Zum Beispiel:

Biotechnologie

Fortschritte bei der Sequenzierungs- und Screening-Technologie haben zu immensen Datenmengen verschiedenster Art geführt, wie zum Beispiel DNA-Sequenzen, Proteinstrukturen, Compound Screens und RNA-Expressions. Techniken des maschinellen Lernens werden ausgiebig auf alle diese Daten angewandt, um Muster herauszufinden, die beim Verstehen biologischer Prozesse helfen können.

e-bol.net

Ermittlung finanzieller Betrügereien

Kreditkartenfirmen sind kontinuierlich auf der Suche nach neuen Wegen, um herauszufinden, ob Transaktionen betrügerisch sind. In diesem Bereich wurden Techniken wie neuronale Netze und induktive Logiken genutzt, um Transaktionen zu überprüfen und unerwünschte Nutzungen abzufangen.

Maschinelles Sehen

Das Interpretieren von Bildern einer Videokamera für militärische Zwecke oder zum Retten von Menschen ist ein aktives Forschungsgebiet. Viele maschinelle Lerntechniken werden verwendet, um zu versuchen, automatisch Eindringlinge aufzuspüren, Fahrzeuge zu identifizieren oder Gesichter zu erkennen. Besonders interessant ist die Verwendung nicht überwachter Techniken, wie die *Independent Component Analysis*, die interessante Merkmale in großen Datenmengen findet.

Produkt-Marketing

Lange Zeit war das Verstehen von Demografie und das Erkennen von Trends mehr eine Kunst als eine Wissenschaft. Neuerdings haben die gewachsenen Möglichkeiten beim Sammeln von Konsumentendaten zu erweiterten Anwendungsbereichen des maschinellen Lernens wie dem Clustering geführt, um die natürliche Aufteilung besser zu verstehen, die in den Märkten vorhanden ist, und um bessere Voraussagen über zukünftige Trends machen zu können.

Lieferkettenoptimierung

Große Unternehmen können Millionen von Dollars sparen, indem sie ihre Lieferketten effektiver machen und die Produktnachfrage in den verschiedenen Bereichen genauer voraussagen können. Die Anzahl der möglichen Lieferkettenvarianten ist riesig, da es viele Faktoren gibt, die die Nachfrage potenziell beeinflussen können. Optimierung und Lerntechniken werden häufig genutzt, um diese Daten zu analysieren.

Aktienmarktanalysen

Seit es einen Aktienmarkt gibt, haben Menschen versucht, mithilfe der Mathematik mehr Geld zu verdienen. Da die Teilnehmer dieses Markts zunehmend fortschrittlichere Techniken einsetzen, wird es notwendig, größere Datenmengen zu analysieren und aufwendigere Methoden zu nutzen, um Muster zu erkennen.

Nationale Sicherheit

Staatliche Institutionen auf der ganzen Welt sammeln große Informationsmengen, und die Analyse dieser Daten erfordert Computer, um Muster zu erkennen und sie mit möglichen Bedrohungsszenarien in Verbindung zu bringen.

Das sind nur ein paar Beispiele, in denen maschinelles Lernen heutzutage intensiv genutzt wird. Da der Trend dahin geht, noch mehr Informationen anzuhäufen, ist es sehr wahrscheinlich, dass weitere Bereiche maschinelles Lernen und statistische

e-bol.net

Techniken nutzen werden, da die Menge an Informationen weit über das hinausgeht, was man ohne entsprechende Hilfe anders verarbeiten könnte.

Aufgrund der Tatsache, dass jeden Tag neue Informationen zur Verfügung gestellt werden, gibt es offensichtlich viele neue Möglichkeiten. Nachdem Sie sich einmal mit ein paar Algorithmen zum maschinellen Lernen vertraut gemacht haben, werden Sie überall Gelegenheiten finden, diese anzuwenden.

e-bol.net

KAPITEL 2

Empfehlungen geben

Um nun mit unserer Reise durch die kollektive Intelligenz zu beginnen, werde ich Ihnen Möglichkeiten zeigen, wie Sie die Vorlieben einer Gruppe von Leuten nutzen können, um anderen Personen Empfehlungen auszusprechen. Es gibt viele Anwendungen für diese Art von Information, zum Beispiel Produktempfehlungen beim Onlineshopping, das Vorschlagen interessanter Websites oder eine Hilfestellung bei der Suche nach Musik und Filmen. Dieses Kapitel zeigt Ihnen, wie Sie ein System aufbauen, mit dem Sie Personen finden, die den gleichen Geschmack haben, um dann automatische Empfehlungen auszusprechen basierend auf den Dingen, die andere Personen mögen.

Sie haben vermutlich schon einmal Empfehlungs-Engines gesehen, wenn Sie zum Beispiel eine Onlineshopping-Site wie Amazon besucht haben. Amazon protokolliert die Kaufgewohnheiten aller ihrer Kunden, und wenn Sie sich an der Site anmelden, verwendet es diese Informationen, um Produkte vorzuschlagen, die Sie vielleicht interessieren. Amazon kann sogar für Sie interessante Filme vorschlagen, wenn Sie bisher nur Bücher gekauft haben. Manche Online-Konzertticketagenturen begutachten die von Ihnen besuchten Konzerte und weisen Sie auf demnächst stattfindende Ereignisse hin, die für Sie von Interesse sein könnten. Sites wie *reddit.com* ermöglichen es Ihnen, Links auf andere Websites zu bewerten, um dann mit Ihren Bewertungen weitere Links vorzuschlagen, die für Sie auch interessant sein könnten.

Anhand dieser Beispiele lässt sich feststellen, dass die Vorlieben der Menschen auf viele unterschiedliche Arten gesammelt werden können. Manchmal handelt es sich bei den Daten um Dinge, die Leute gekauft haben, und Meinungen zu diesen Dingen, die als Ja/Nein-Stimmen oder als Bewertung von eins bis fünf dargestellt werden. In diesem Kapitel werden wir verschiedene Möglichkeiten dazu betrachten, wie man diese Fälle so repräsentiert, dass sie alle mit den gleichen Algorithmen verarbeitet werden können, und wir werden funktionsfähige Beispiele mit Filmkritiken und Social Bookmarking-Daten erstellen.

e-bol.net

Kollaboratives Filtern

Wenn Sie ohne aufwendige Technik Empfehlungen für Produkte, Filme oder nette Websites erhalten wollen, können Sie Ihre Freunde fragen. Sie wissen auch, dass manche Ihrer Freunde Ihren Geschmack besser trafen als andere – das haben Sie gelernt, indem Sie eine ganze Weile beobachteten, wer normalerweise die gleichen Dinge wie Sie bevorzugt. Wenn es mehr und mehr Auswahlmöglichkeiten gibt, wird es immer schwieriger zu entscheiden, was Sie wollen, wenn Sie nur eine kleine Gruppe von Leuten befragen, da dieser Gruppe eventuell nicht alle Möglichkeiten bekannt sind. Das ist der Grund, warum eine Reihe von Techniken mit dem Namen *kollaboratives Filtern* (*Collaborative Filtering*) entwickelt wurde.

Ein kollaborativer Filteralgorithmus funktioniert normalerweise so, dass er eine große Gruppe von Personen durchsucht und eine kleinere Untergruppe findet, die Ihren Geschmack teilt. Er schaut sich dann die anderen Dinge an, die die Leute dieser Gruppe mögen, und erstellt eine bewertete Liste mit Empfehlungen. Es gibt viele verschiedene Möglichkeiten zu entscheiden, welche Leute ähnlich handeln und wie ihre Vorlieben zu einer Liste zu kombinieren sind – dieses Kapitel wird ein paar davon behandeln.

Der Begriff *kollaboratives Filtern* wurde als Erstes von David Goldberg bei Xerox PARC im Jahr 1992 in dem Text »Using collaborative filtering to weave an information tapestry« verwendet. Er entwarf ein System namens *Tapestry*, das es den Leuten ermöglichte, Dokumente entweder als interessant oder als uninteressant zu kennzeichnen, und das diese Information verwendete, um Dokumente für andere Personen auszuwählen.

Es gibt mittlerweile Hunderte von Websites, die irgendeine Form von kollaborativem Filteralgorithmus für Filme, Musik, Bücher, Dates, Shopping, andere Websites, Podcasts, Artikel und selbst Witze verwenden.

Vorlieben sammeln

Das Erste, was Sie brauchen, ist ein Weg, die verschiedenen Menschen mit ihren Vorlieben zu repräsentieren. In Python ist eine sehr einfache Möglichkeit dafür die Verwendung eines *verschachtelten Dictionaries*. Wenn Sie die Beispiele in diesem Abschnitt ausprobieren wollen, erstellen Sie eine Datei *recommendations.py* und fügen den folgenden Code ein, um die Daten zu erzeugen:

```
# ein Dictionary mit Filmkritiken und deren Bewertungen
# von ein paar Filmen
critics={'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5,
 'Just My Luck': 3.0, 'Superman Returns': 3.5, 'You, Me and Dupree': 2.5,
 'The Night Listener': 3.0},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5,
```

e-bol.net

```
 'Just My Luck': 1.5, 'Superman Returns': 5.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 3.5},
'Michael Phillips': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.0,
 'Superman Returns': 3.5, 'The Night Listener': 4.0},
'Claudia Puig': {'Snakes on a Plane': 3.5, 'Just My Luck': 3.0,
 'The Night Listener': 4.5, 'Superman Returns': 4.0,
 'You, Me and Dupree': 2.5},
'Mick LaSalle': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'Just My Luck': 2.0, 'Superman Returns': 3.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 2.0},
'Jack Matthews': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'The Night Listener': 3.0, 'Superman Returns': 5.0, 'You, Me and Dupree': 3.5},
'Toby': {'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman Returns':4.0}}
```

Sie werden mit Python in diesem Kapitel interaktiv arbeiten, daher sollten Sie *recommendations.py* dort speichern, wo der interaktive Interpreter von Python es finden kann. Das kann im Verzeichnis *python/Lib* sein, aber am einfachsten ist es, den Python-Interpreter im gleichen Verzeichnis zu starten, in dem Sie die Datei gespeichert haben.

Dieses Dictionary nutzt eine Bewertung von 1 bis 5 als Ausdruck dafür, wie sehr die Filmkritiker (und ich) einen gegebenen Film mochten. Egal wie Vorlieben ausgedrückt werden – Sie brauchen eine Möglichkeit, sie in numerische Werte umzuwandeln. Wenn Sie eine Shopping-Site gebaut haben, verwenden Sie vielleicht den Wert 1, um anzugeben, dass jemand in der Vergangenheit einen Gegenstand gekauft hat, während 0 dafür steht, dass er es nicht tat. Bei einer Site, auf der die Leute Nachrichten bewerten, könnten die Werte –1, 0 und 1 für »ablehnend«, »keine Wertung« und »zustimmend« dienen, wie in Tabelle 2-1 gezeigt.

Tabelle 2-1: Mögliche Abbildungen von Benutzeraktionen auf numerische Werte

Konzerttickets		Onlineshopping		Site- Empfehlung	
Gekauft	1	Gekauft	2	Zustimmend	1
Nicht gekauft	0	Angeschaut	1	Keine Wertung	0
		Nicht gekauft	0	Ablehnend	–1

Die Verwendung eines Dictionary ist beim Experimentieren mit den Algorithmen und zum Demonstrieren sehr praktisch. Es lässt sich leicht durchsuchen und anpassen. Starten Sie Ihren Python-Interpreter und testen Sie ein paar Befehle aus:

```
c:\code\collective\chapter2> python
Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>> from recommendations import critics
>> critics['Lisa Rose']['Lady in the Water']
2.5
>> critics['Toby']['Snakes on a Plane']=4.5
>> critics['Toby']
{'Snakes on a Plane':4.5,'You, Me and Dupree':1.0}
```

e-bol.net

Auch wenn Sie eine große Zahl von Vorlieben in einem Dictionary im Speicher unterbringen können, werden Sie für sehr große Datenmengen vermutlich eher eine Datenbank nutzen wollen.

Ähnliche Benutzer finden

Nach dem Sammeln von Daten über das, was die Leute mögen, brauchen Sie jetzt eine Möglichkeit zu bestimmen, wie ähnlich sich diese in ihrem Geschmack sind. Das tun Sie, indem Sie jede Person mit jeder anderen vergleichen und einen *Similarity Score* berechnen. Dazu gibt es verschiedene Wege. In diesem Abschnitt werde ich Ihnen zwei Systeme zum Berechnen von Similarity Scores zeigen: den *euklidischen Abstand* und die *Pearson-Korrelation*.

Euklidischer Abstand

Ein sehr einfacher Weg, einen Similarity Score zu berechnen, ist die Verwendung des euklidischen Abstands, der die von den Personen zusammen bewerteten Elemente nimmt und sie als Achsen eines Graphen nutzt. Sie können dann die Personen im Graphen eintragen und sehen, wie nahe beieinander sie sich befinden (siehe Abbildung 2-1).

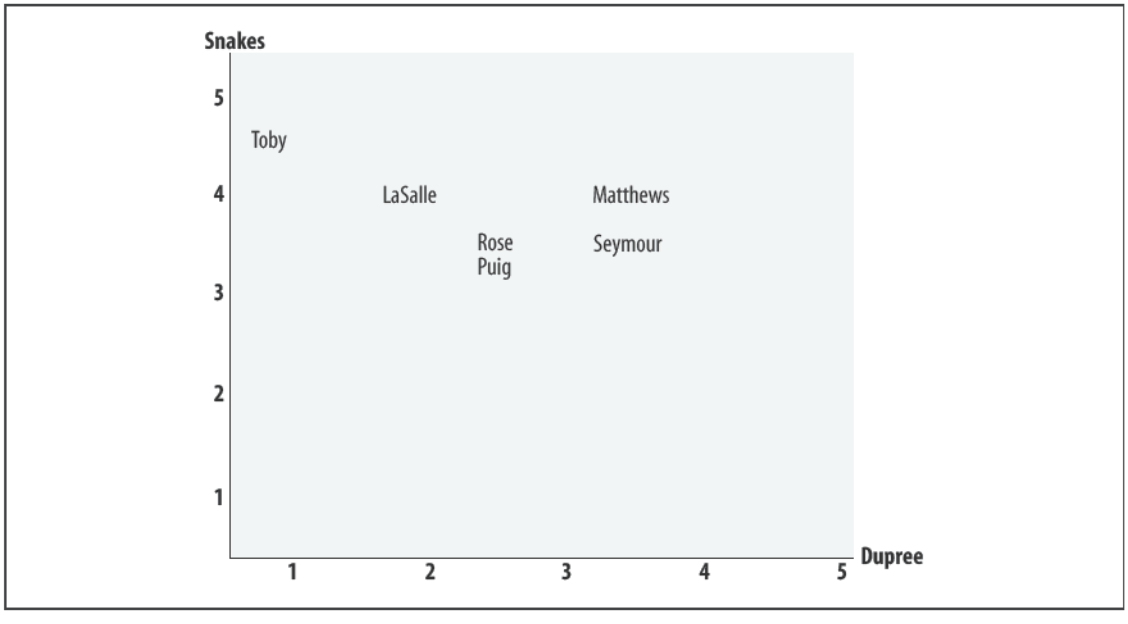

Abbildung 2-1: Personen im Preference Space

Diese Grafik zeigt die Leute im *Preference Space*. Toby wurde auf der Snakes-Achse bei 4,5 und auf der Dupree-Achse bei 1,0 eingetragen. Je näher sich zwei Personen im Preference Space sind, desto ähnlicher sind ihre Vorlieben. Da der Graph zweidimensional ist, können Sie immer nur auf zwei Bewertungen zugleich schauen, aber das Prinzip gilt genauso für größere Bewertungssets.

e-bol.net

Um den Abstand zwischen Toby und LaSalle im Graphen zu errechnen, nehmen Sie den Abstand auf jeder Achse, quadrieren ihn und addieren diese Werte, dann ziehen Sie aus der Summe die Wurzel. In Python können Sie die Funktion `pow(n,2)` zum Quadrieren einer Zahl verwenden und die Wurzel mit der Funktion `sqrt` ziehen:

```
>> from math import sqrt
>> sqrt(pow(5-4,2)+pow(4-1,2))
3.1622776601683795
```

Diese Formel berechnet den Abstand, der umso kleiner ist, je ähnlicher sich zwei Personen sind. Sie brauchen aber eine Funktion, die einen größeren Wert für diejenigen ausgibt, die sich ähnlicher sind. Das können Sie erreichen, indem Sie zur Funktion 1 addieren (damit Sie keinen Fehler bei einer Division durch null erhalten) und sie invertieren:

```
>> 1/(1+sqrt(pow(5-4,2)+pow(4-1,2)))
0.2402530733520421
```

Diese neue Funktion liefert immer einen Wert zwischen 0 und 1 zurück, wobei 1 bedeutet, dass zwei Personen identische Vorlieben haben. Sie können nun alles zusammenbringen, um eine Funktion zum Berechnen der Ähnlichkeit zu erstellen. Fügen Sie der Datei *recommendations.py* den folgenden Code hinzu:

```
from math import sqrt

# Liefert einen abstandsbasierten Similarity Score für
# person1 und person2 zurück.
def sim_distance(prefs,person1,person2):
  # Liste der gemeinsamen Elemente bestimmen.
  si={}
  for item in prefs[person1]:
    if item in prefs[person2]:
       si[item]=1

  # Wenn keine gemeinsamen Bewertungen existiert, 0 zurückgeben.
  if len(si)==0: return 0

  # Quadrate aller Abstände addieren.
  sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
                      for item in prefs[person1] if item in prefs[person2]])

  return 1/(1+*sqr*t(sum_of_squares))
```

Diese Funktion kann nun mit zwei Namen aufgerufen werden, um einen Similarity Score zu erhalten. Rufen Sie in Ihrem Python-Interpreter Folgendes auf:

```
>>> reload(recommendations)
>>> recommendations.sim_distance(recommendations.critics,
...   'Lisa Rose','Gene Seymour')
0.148148148148
```

e-bol.net

Damit erhalten Sie einen Similarity Score zwischen Lisa Rose und Gene Seymour. Versuchen Sie es mit anderen Namen, um herauszubekommen, ob Sie Personen finden können, die mehr oder weniger gemeinsam haben.

Pearson-Korrelationskoeffizient

Ein etwas ausgefeilterer Weg, um die Ähnlichkeit zwischen den Interessen von Personen zu bestimmen, ist die Verwendung des Pearson-Korrelationskoeffizienten. Der Korrelationskoeffizient ist ein Wert, der bestimmt, wie gut zwei Datenmengen auf eine gerade Linie passen. Die Formel ist komplizierter als der euklidische Abstand, liefert aber meist bessere Ergebnisse in Situationen, in denen die Daten nicht normalisiert sind – zum Beispiel wenn die Filmbewertungen im Schnitt eher harsch als durchschnittlich sind.

Um sich ein echtes Bild von dieser Methode zu machen, können Sie die Bewertungen zweier Kritiker in einen Graphen eintragen, wie in Abbildung 2-2 gezeigt. *Superman* wurde von Mick LaSalle mit 3 und von Gene Seymour mit 5 bewertet, daher wird er im Graphen bei (3;5) platziert.

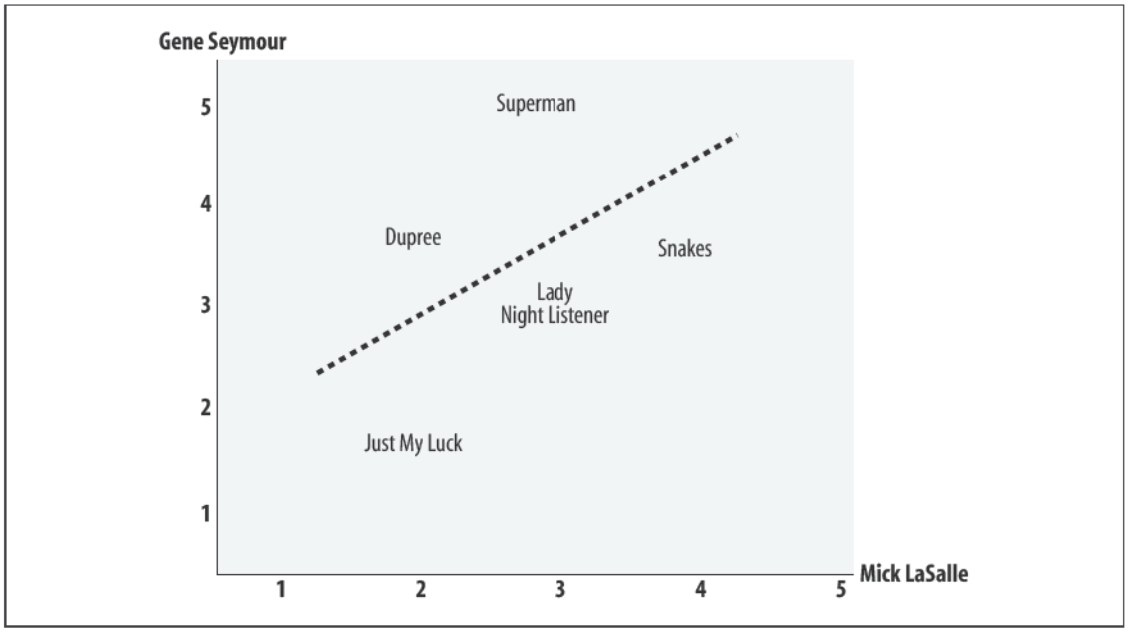

Abbildung 2-2: Der Vergleich zweier Filmkritiker mit einer Punktwolke

Sie können im Graphen auch eine Gerade sehen. Diese wird als *Ausgleichsgerade* bezeichnet, da sie den Punkten im Graphen so nahe kommt wie möglich. Wären die Bewertungen der beiden Kritiker für jeden Film identisch, würde diese Gerade diagonal sein und jeden eingetragenen Punkt im Graphen durchlaufen, womit man einen perfekten Korrelationskoeffizienten von 1 hätte. Im hier gezeigten Fall sind die Kritiker bei manchen Filmen nicht einer Meinung, daher liegt der Korrelations-

e-bol.net

koeffizient bei ungefähr 0,4. Abbildung 2-3 zeigt ein Beispiel einer deutlich höheren Korrelation mit dem ungefähren Wert 0,75.

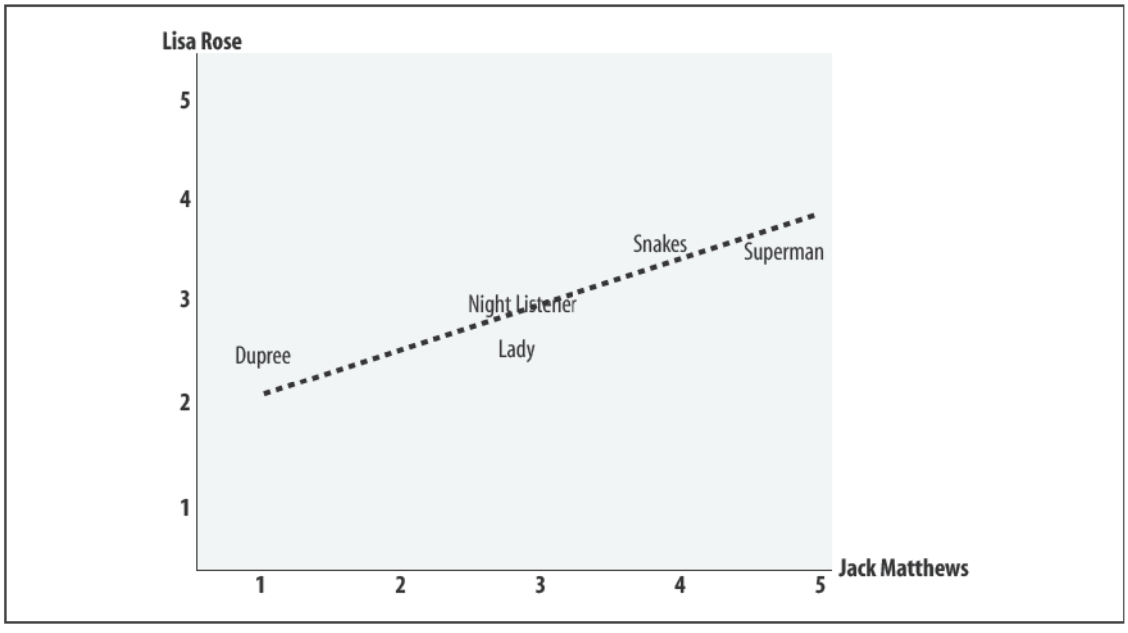

Abbildung 2-3: Zwei Kritiker mit einem hohen Korrelationskoeffizienten

Ein interessanter Aspekt bei der Verwendung des Korrelationskoeffizienten, den Sie im Graphen erkennen können, ist ein Ausgleich der *Grade Inflation*, also zu »guter« Bewertungen. In diesem Graphen tendiert Jack Matthews dazu, bessere Bewertungen zu geben als Lisa Rose, aber die Gerade passt trotzdem, da sie relativ gesehen ähnliche Vorlieben haben. Wenn ein Kritiker höhere Bewertungen vergibt als der andere, kann die Korrelation immer noch perfekt sein, wenn der Unterschied zwischen den Wertungen konsistent ist. Der weiter oben beschriebene euklidische Abstand wird hier sagen, dass zwei Kritiker nicht ähnlich sind, weil einer durchgehend extremer bewertet als der andere, selbst wenn ihre Geschäcker sehr ähnlich sind. Je nachdem, wie Ihre Anwendung aussieht, kann dieses Verhalten erwünscht sein oder auch nicht.

Der Code für den Korrelationskoeffizienten sucht zuerst nach Filmen, die von beiden Kritikern bewertet wurden. Dann berechnet er die Summe und die Quadratsumme der Bewertungen für die beiden Kritiker, ebenso die Summe der Produkte ihrer Bewertungen. Schließlich nutzt er diese Werte, um den Korrelationskoeffizienten zu ermitteln, der unten in Fettdruck zu sehen ist. Anders als bei der Abstandsmessung ist diese Formel nicht sehr intuitiv, sie teilt Ihnen allerdings mit, wie sehr sich die Variablen zusammen verändern, geteilt durch das Produkt der individuellen Veränderungen.

Um diese Formel zu nutzen, erstellen Sie eine neue Funktion mit der gleichen Signatur wie bei `sim_distance` in *recommendations.py*:

e-bol.net

```
# Liefert den Korrelationskoeffizienten für p1 und p2.
def sim_pearson(prefs,p1,p2):
  # Liste der gemeinsam bewerteten Elemente
  si={}
  for item in prefs[p1]:
    if item in prefs[p2]: si[item]=1

  # Anzahl der Elemente
  n=len(si)

  # Wenn es keine gemeinsamen Wertungen gibt, gib 0 zurück.
  if n==0: return 0

  # Bewertungen summieren.
  sum1=sum([prefs[p1][it] for it in si])
  sum2=sum([prefs[p2][it] for it in si])

  # Quadrate summieren.
  sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
  sum2Sq=sum([pow(prefs[p2][it],2) for it in si])

  # Produkte summieren.
  pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

  # Korrelationskoeffizient berechnen.
  num=pSum-(sum1*sum2/n)
  den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
  if den==0: return 0

  r=num/den

  return r
```

Diese Funktion wird einen Wert zwischen –1 und 1 zurückgeben. Ein Wert von 1 bedeutet, dass die beiden Personen genau die gleichen Bewertungen für jedes Objekt haben. Anders als bei der Abstandsmessung müssen Sie diesen Wert nicht ändern, um die richtige Skala zu erhalten. Sie können nun versuchen, den Korrelationskoeffizienten für Abbildung 2-3 zu erhalten:

```
>>> reload(recommendations)
>>> print recommendations.sim_pearson(recommendations.critics,
...  'Lisa Rose','Gene Seymour')
0.396059017191
```

Welchen Similarity Score sollten Sie nutzen?

Ich habe Funktionen für zwei verschiedene Metriken präsentiert, aber es gibt natürlich noch mehr Möglichkeiten, die Ähnlichkeit zwischen zwei Datenmengen zu messen. Es hängt von Ihrer Anwendung ab, welche davon die beste ist, und es lohnt sich, den Korrelationskoeffizienten, den euklidischen Abstand oder andere auszuprobieren, um zu sehen, woher die besten Ergebnisse kommen.

e-bol.net

Die Funktionen im Rest dieses Kapitels haben einen optionalen Parameter `similarity`, der auf eine Funktion zeigt, um es beim Experimentieren einfacher zu machen: Geben Sie `sim_pearson` oder `sim_vector` an, um auszuwählen, welcher Ähnlichkeitswert zu nutzen ist. Es gibt noch viele andere Funktionen, wie den *Jaccard-Koeffizienten* oder die *Manhattan-Metrik*, die Sie in einer Ähnlichkeitsfunktion nutzen können, solange sie die gleiche Signatur besitzt und eine Gleitkommazahl zurückgibt, bei der ein höherer Wert für mehr Ähnlichkeit steht.

Sie können mehr über andere Metriken für das Vergleichen von Elementen unter *http://en.wikipedia.org/wiki/Metric_%28mathematics%29#Examples* beziehungsweise *http://de.wikipedia.org/wiki/Metrischer_Raum#Beispiele:_durch_Normen_erzeugte_Metriken* lesen.

Bewerten der Kritiker

Nachdem Sie nun Funktionen zum Vergleichen zweier Personen haben, können Sie eine Funktion erstellen, die jeden mit einer gegebenen Person vergleicht und die beste Übereinstimmung findet. In diesem Fall bin ich daran interessiert zu erfahren, welche Filmkritiker ähnliche Vorlieben wie ich hat, sodass ich weiß, wessen Rat ich folgen kann, wenn ich mich für einen Film entscheiden will. Fügen Sie diese Funktion der Datei *recommendations.py* hinzu, um eine sortierte Liste der Personen mit ähnlichen Vorlieben wie die der angegebenen `person` zu erhalten:

```
# Liefert die besten Übereinstimmungen für person aus dem
# Dictionary prefs zurück.
# Anzahl der Ergebnisse und die similarity-Funktion sind optional.
def topMatches(prefs,person,n=5,similarity=sim_pearson):
  scores=[(similarity(prefs,person,other),other)
                 for other in prefs if other!=person]

  # Sortieren der Liste, damit der höchste Wert oben ist.
  scores.sort( )
  scores.reverse( )
  return scores[0:n]
```

Diese Funktion verwendet eine *List Comprehension* in Python, um mich mit jedem anderen Benutzer im Dictionary mithilfe einer der vorher definierten Abstandsfunktionen zu vergleichen. Dann werden die ersten *n* der sortierten Ergebnisse zurückgegeben.

Ein Aufruf dieser Funktion mit meinem eigenen Namen liefert mir eine Liste der Filmkritiken und ihrer Similarity Scores:

```
>> reload(recommendations)
>> recommendations.topMatches(recommendations.critics,'Toby',n=3)
[(0.99124070716192991, 'Lisa Rose'), (0.92447345164190486, 'Mick LaSalle'),
 (0.89340514744156474, 'Claudia Puig')]
```

e-bol.net

Aus diesem Ergebnis kann ich ablesen, dass ich die Kritiken von Lisa Rose lesen sollte, da ihr Geschmack meinem recht ähnlich zu sein scheint. Sollten Sie von diesen Filmen welche gesehen haben, können Sie versuchen, sich selbst dem Dictionary hinzuzufügen, um zu sehen, wer Ihr bevorzugter Kritiker sein kann.

Dinge empfehlen

Es ist toll, wenn man einen guten Kritiker findet, dessen Texte man gern genauer liest, aber was ich wirklich will, ist genau jetzt eine Empfehlung für einen Film. Ich könnte einfach die Bewertungen der Person anschauen, die meinen Geschmack annähernd trifft, um dort einen Film zu finden, den sie mag und den ich noch nicht gesehen habe, aber das wäre zu unsicher. Solch ein Vorgehen könnte unbeabsichtigt Kritiker anzeigen, die manche der von mir bevorzugten Filme nicht bewertet haben. Auch könnte ein Kritiker zurückgegeben werden, der seltsamerweise einen Film liebt, der von allen anderen Kritikern, die `topMatches` zurückgibt, schlecht bewertet wurde.

Um dieses Problem zu lösen, müssen Sie die Elemente bewerten, indem Sie eine gewichtete Bewertung nutzen, die für eine Ordnung der Kritiken sorgt. Nehmen Sie die Bewertungen aller anderen Kritiker und multiplizieren Sie sie mit dem Similarity Score zu mir. Tabelle 2-2 zeigt, wie das funktioniert.

Tabelle 2-2: Empfehlungen für Toby erzeugen

Kritiker	Similarity Score	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0,99	3,0	2,97	2,5	2,48	3,0	2,97
Seymour	0,38	3,0	1,14	3,0	1,14	1,5	0,57
Puig	0,89	4,5	4,02			3,0	2,68
LaSalle	0,92	3,0	2,77	3,0	2,77	2,0	1,85
Matthews	0,66	3,0	1,99	3,0	1,99		
Gesamt			12,89		8,38		8,07
Summe Sim.			3,84		2,95		3,18
Gesamt/Summe Sim.			3,35		2,83		2,53

Diese Tabelle zeigt die Korrelationskoeffizienten für jeden Kritiker und die Bewertungen, die die Kritiker für die drei Filme vergeben haben (*The Night Listener*, *Lady in the Water* und *Just My Luck*), die ich selbst bewertet habe. Die Spalten, die mit S.x beginnen, geben den Similarity Score multipliziert mit der Bewertung an, sodass eine Person, deren Geschmack meinem ähnelt, zur Gesamtwertung mehr beitragen wird als eine Person, die sich von mir stark unterscheidet. Die Zeile »Gesamt« gibt die Summe aller dieser Produkte an.

e-bol.net

Sie könnten einfach diese Summe nutzen, um die Bewertungen zu berechnen, aber dann würde ein Film, der von mehr Leuten bewertet wurde, einen großen Vorteil haben. Um das auszugleichen, müssen Sie durch die Summe aller Similarity Scores der Kritiker teilen, die diesen Film bewertet haben (die Zeile »Summe Sim.« in der Tabelle). Da *The Night Listener* von jedem bewertet wurde, wird seine Summe durch die Summe aller Similarity Scores geteilt. *Lady in the Water* wurde nicht von Puig bewertet, daher wird die Bewertung dieses Films nur durch die Summe aller anderen Similarity Scores geteilt. Die letzte Zeile zeigt die Ergebnisse dieser Divisionen.

Der Code dafür ist ziemlich einfach und funktioniert sowohl mit dem euklidischen Abstand als auch mit dem Pearson-Korrelationskoeffizienten. Fügen Sie ihn der Datei *recommendations.py* hinzu:

```
# Empfehlungen für eine Person erhalten, indem ein gewichteter
# Durchschnitt aller anderen Bewertungen gebildet wird.
def getRecommendations(prefs,person,similarity=sim_pearson):
  totals={}
  simSums={}
  for other in prefs:
    # Nicht mit mir selbst vergleichen.
    if other==person: continue
    sim=similarity(prefs,person,other)

    # Werte <= 0 ignorieren.
    if sim<=0: continue
    for item in prefs[other]:

      # Nur Filme bewerten, die ich noch nicht gesehen habe.
      if item not in prefs[person] or prefs[person][item]==0:
        # Similarity * Bewertung
        totals.setdefault(item,0)
        totals[item]+=prefs[other][item]*sim
        # Summe der Similarities
        simSums.setdefault(item,0)
        simSums[item]+=sim

  # Normalisierte Liste erstellen.
  rankings=[(total/simSums[item],item) for item,total in totals.items()]

  # Sortierte Liste zurückgeben.
  rankings.sort()
  rankings.reverse()
  return rankings
```

Dieser Code läuft durch alle anderen Personen im Dictionary `prefs` durch. Für jede Person wird jeder Film durchlaufen, der bewertet wurde. Die fett gedruckte Zeile zeigt, wie die endgültige Bewertung ermittelt wird – die Bewertung für jeden Film wird mit dem Similarity Score multipliziert, und diese Produkte werden dann addiert. Schließlich werden die Zahlen normalisiert, indem sie durch die entsprechende Summe der Ähnlichkeiten geteilt werden, um dann eine sortierte Liste zurückzugeben.

e-bol.net

Jetzt können Sie herausfinden, welche Filme ich als Nächstes anschauen sollte:

```
>>> reload(recommendations)
>>> recommendations.getRecommendations(recommendations.critics,'Toby')
[(3.3477895267131013, 'The Night Listener'), (2.8325499182641614, 'Lady in the
Water'), (2.5309807037655645, 'Just My Luck')]
>>> recommendations.getRecommendations(recommendations.critics,'Toby',
...   similarity=recommendations.sim_distance)
[(3.5002478401415877, 'The Night Listener'), (2.7561242939959363, 'Lady in the
Water'), (2.4619884860743739, 'Just My Luck')]
```

Sie erhalten nicht nur eine sortierte Liste der Filme, sondern auch noch eine Abschätzung über meine Bewertung jedes einzelnen Films. Anhand dieser Liste kann ich entscheiden, ob ich einen Film überhaupt schauen möchte oder lieber etwas ganz anderes tue. Je nach Ihrer Anwendung können Sie sich entscheiden, gar keine Empfehlung zu geben, wenn es nichts gibt, was den Standards eines bestimmten Benutzers entspricht. Sie werden feststellen, dass die Ergebnisse nur wenig durch die Wahl der Ähnlichkeitsmetrik beeinflusst werden.

Sie haben nun ein vollständiges Empfehlungssystem aufgebaut, das mit jeder Art von Produkt oder Link funktioniert. Sie müssen nur ein Dictionary mit Personen, Objekten und Bewertungen füllen und können dann damit Empfehlungen für jede der Personen abgeben. Weiter unten in diesem Kapitel werden Sie sehen, wie Sie die API von *del.icio.us* nutzen können, um echte Daten für das Empfehlen von Websites zu erhalten.

Produkte finden

Nun wissen Sie, wie Sie ähnliche Personen finden und einer bestimmten Person Produkte empfehlen können. Was aber, wenn Sie herausfinden wollen, welche Produkte zueinander passen? Sie haben das vielleicht schon auf Shopping-Websites gesehen, insbesondere dann, wenn die Site nicht viele Daten über Sie hat. Ein Ausschnitt aus der Webseite von Amazon für das Buch *Programming Python* ist in Abbildung 2-4 zu sehen.

Customers who bought this item also bought

Learning Python, Second Edition by Mark Lutz
Python Cookbook by Alex Martelli
Python in a Nutshell by Alex Martelli
Python Essential Reference (2nd Edition) by David Beazley
Foundations of Python Network Programming (Foundations) by John Goerzen

Explore similar items : Books (42)

Abbildung 2-4: Amazon zeigt Produkte, die »Programming Python« ähneln

e-bol.net

In diesem Fall können Sie die Ähnlichkeit bestimmen, indem Sie begutachten, wer ein bestimmtes Produkt mochte, und dann schauen, welche anderen Produkte diejenigen bevorzugten. Das ist tatsächlich die gleiche Methode, die wir schon weiter oben benutzt haben, um die Ähnlichkeit zwischen Personen zu bestimmen – Sie müssen nur die Personen und die Objekte miteinander vertauschen. Nutzen Sie also die gleichen Methoden wie oben, indem Sie das Dictionary entsprechend von:

```
{'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}
```

umwandeln in:

```
{'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}} etc...
```

Fügen Sie der *recommendations.py* diesen Code hinzu, um die Transformation zu ermöglichen:

```
def transformPrefs(prefs):
  result={}
  for person in prefs:
    for item in prefs[person]:
      result.setdefault(item,{})

      # Personen und Objekte vertauschen.
      result[item][person]=prefs[person][item]
  return result
```

Und jetzt rufen Sie die Funktion `topMatches` auf, die schon vorher verwendet wurde, um die Filme zu finden, die am besten zu *Superman Returns* passen:

```
>> reload(recommendations)
>> movies=recommendations.transformPrefs(recommendations.critics)
>> recommendations.topMatches(movies,'Superman Returns')
[(0.657, 'You, Me and Dupree'), (0.487, 'Lady in the Water'), (0.111, 'Snakes on a
Plane'), (-0.179, 'The Night Listener'), (-0.422, 'Just My Luck')]
```

Beachten Sie, dass es in diesem Beispiel tatsächlich auch negative Korrelationskoeffizienten gibt, die zeigen, dass diejenigen, die *Superman Returns* gut finden, den Film *Just My Luck* eher nicht mögen werden, wie in Abbildung 2-5 zu sehen ist.

Um alles noch mehr umzuordnen, können Sie empfehlenswerte Kritiker für einen Film generieren. Vielleicht versuchen Sie herauszubekommen, wen man zu einer Premiere einladen kann:

```
>> recommendations.getRecommendations(movies,'Just My Luck')
[(4.0, 'Michael Phillips'), (3.0, 'Jack Matthews')]
```

Das Austauschen von Personen und Objekten führt nicht zwangsläufig zu sinnvollen Ergebnissen, aber in vielen Fällen werden Ihnen dadurch interessante Vergleiche ermöglicht. Ein Onlinehändler sammelt vielleicht die Bestellhistorien, um seinen Kunden Produkte zu empfehlen. Vertauscht man die Produkte mit den Personen,

e-bol.net

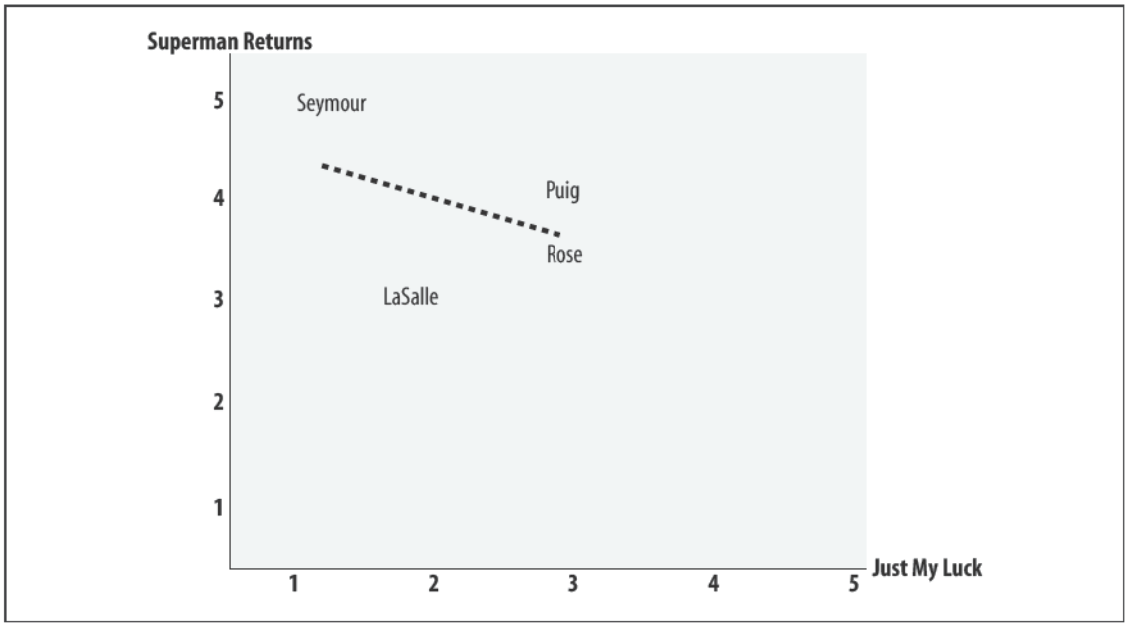

Abbildung 2-5: Superman Returns und Just My Luck haben eine negative Korrelation

wie es hier getan wurde, erhält man die Möglichkeit, nach Personen zu suchen, die bestimmte Produkte kaufen würden. Das könnte sehr sinnvoll sein, wenn man eine Marketing-Kampagne starten will, um bestimmte Produkte abzuverkaufen. Eine andere mögliche Verwendung läge darin sicherzustellen, dass neue Links bei einer Empfehlungsseite für Links von den Personen bemerkt werden, die sie möglicherweise interessant finden könnten.

Link-Empfehlungen mit del.icio.us erstellen

Dieser Abschnitt zeigt Ihnen, wie Sie Daten von einer der beliebtesten Online-Bookmark-Sites laden, mit diesen Daten passende Benutzer finden und Links empfehlen können, die diese noch nicht gesehen haben. Diese Site, die Sie über *http://del.icio.us* erreichen, ermöglicht es den Leuten, ein Konto einzurichten und Links zu posten, die sie später wieder ansurfen wollen. Sie können die Site besuchen und sich Links anzeigen lassen, die andere Personen gepostet haben, aber auch »beliebte« Links ausgeben lassen, die von vielen verschiedenen Leuten gepostet wurden. Eine Beispielseite von *del.icio.us* ist in Abbildung 2-6 zu sehen.

Im Gegensatz zu manchen anderen Sites, auf denen man Links veröffentlichen kann, gibt es bei *del.icio.us* (zumindest zum Zeitpunkt der Entstehung dieses Buchs) keine Möglichkeit, Personen mit ähnlichem Geschmack zu finden oder sich Links empfehlen zu lassen. Glücklicherweise können Sie die in diesem Kapitel besprochenen Techniken nutzen, um die Funktionalität selbst hinzuzufügen.

e-bol.net

Abbildung 2-6: Die Popular-Seite zum Thema Programming bei del.icio.us

Die API von del.icio.us

Daten von *del.icio.us* werden über eine API im XML-Format bereitgestellt. Um das Ganze für Sie noch einfacher zu machen, gibt es sogar eine Python-API, die Sie unter *http://code.google.com/p/pydelicious/source* oder *http://oreilly.com/catalog/9780596529321* herunterladen können.

Damit Sie die Beispiele in diesem Abschnitt selbst testen können, müssen Sie die neueste Version dieser Bibliothek herunterladen und in den *library*-Pfad von Python kopieren. (In Anhang A finden Sie mehr Informationen zur Installation dieser Bibliothek.)

Diese Bibliothek hat viele einfache Aufrufmöglichkeiten, um Links zu erhalten, die dort eingetragen wurden. Um zum Beispiel eine Liste der neuesten beliebtesten Links zum Thema Programmierung zu erhalten, können Sie die Funktion `get_popular` nutzen:

```
>> import pydelicious
>> pydelicious.get_popular(tag='programming')
[{'count': '', 'extended': '', 'hash': '', 'description': u'How To Write
Unmaintainable Code', 'tags': '', 'href': u'http://thc.segfault.net/root/phun/
unmaintain.html', 'user': u'dorsia', 'dt': u'2006-08-19T09:48:56Z'}, {'count': '',
'extended': '', 'hash': '', 'description': u'Threading in C#', 'tags': '', 'href':
u'http://www.albahari.com/threading/', 'user': u'mmihale', 'dt': u'2006-05-17T18:
09:24Z'},
...etc...
```

Sie können sehen, dass eine Liste mit Dictionaries zurückgegeben wird, von denen jedes eine URL, eine Beschreibung und den Benutzer enthält, der die URL gepostet hat. Da Sie mit Livedaten arbeiten, werden Ihre Ergebnisse anders aussehen. Es gibt zwei weitere Funktionen, die Sie nutzen sollten – `get_urlposts` für alle Posts einer gegebenen URL und `get_userposts` für alle Posts eines gegebenen Benutzers. Die Daten dieser Funktionen werden auf die gleiche Art und Weise zurückgegeben.

e-bol.net

Erstellen der Daten

Es ist nicht möglich, sämtliche Benutzer-Posts von *del.icio.us* herunterzuladen, daher müssen Sie sich für eine Untermenge entscheiden. Sie können das machen, wie Sie wollen, aber fürinteressante Ergebnisse in diesem Beispiel ist es besser, Personen zu finden, die häufig Einträge vornehmen und ein paar ähnliche Posts haben.

Eine Möglichkeit dafür ist es, eine Liste der Benutzer zu laden, die in letzter Zeit einen beliebten Link mit einem bestimmten Tag gepostet haben. Erstellen Sie eine neue Datei mit dem Namen *deliciousrec.py* und geben Sie den folgenden Code ein:

```
from pydelicious import get_popular,get_userposts,get_urlposts

def initializeUserDict(tag,count=5):
  user_dict={}
  # Die count beliebtesten Posts holen.
  for p1 in get_popular(tag=tag)[0:count]:
    # Alle Benutzer finden, die dies gepostet haben.
    for p2 in get_urlposts(p1['href']):
      user=p2['user']
      user_dict[user]={}
  return user_dict
```

Damit erhalten Sie ein Dictionary mit ein paar Benutzern, die alle auf ein leeres Dictionary verweisen, das darauf wartet, mit Links gefüllt zu werden. Die API liefert nur die 30 letzten Personen zurück, die den Link gepostet haben, daher sammelt die Funktion die Benutzer aus den ersten fünf Links, um eine größere Datenmenge aufzubauen.

Anders als bei den Daten mit den Filmkritikern gibt es in diesem Fall nur zwei mögliche Bewertungen: 0, wenn der Benutzer diesen Link nicht gepostet hat, und 1, wenn er es getan hat. Mit der API können Sie nun eine Funktion erstellen, um die Bewertungen all dieser Benutzer einzutragen. Fügen Sie diesen Code der Datei *deliciousrec.py* hinzu:

```
def fillItems(user_dict):
  all_items={}
  # Links finden, die von allen Benutzern gepostet sind.
  for user in user_dict:
    for i in range(3):
      try:
        posts=get_userposts(user)
        break
      except:
        print "Problem mit Benutzer "+user+", neuer Versuch"
        time.sleep(4)
    for post in posts:
      url=post['href']
      user_dict[user][url]=1.0
      all_items[url]=1
```

e-bol.net

```
# Fehlende Einträge mit 0 auffüllen.
for ratings in user_dict.values():
  for item in all_items:
    if item not in ratings:
      ratings[item]=0.0
```

Das kann genutzt werden, um eine Datenmenge aufzubauen, die denen des Dictionary `critics` ähnelt, das Sie per Hand am Anfang dieses Kapitels angelegt haben:

```
>> from deliciousrec import *
>> delusers=initializeUserDict('programming')
>> delusers ['tsegaran']={} # Fügen Sie sich selbst hinzu, wenn Sie
                            # delicious nutzen.
>> fillItems(delusers)
```

Die dritte Zeile fügt den Benutzer `tsegaran` der Liste hinzu. Sie können `tsegaran` durch Ihren eigenen Benutzernamen ersetzen, wenn Sie *del.icio.us* verwenden.

Der Aufruf von `fillItems` kann ein paar Minuten dauern, da er ein paar Hundert Requests zur Site schickt. Manchmal blockiert die API Requests, die zu häufig wiederholt werden. In diesem Fall macht der Code eine Pause und versucht bis zu drei Mal, den Benutzer herunterzuladen.

Nachbarn und Links empfehlen

Nachdem Sie sich nun Daten beschafft haben, können Sie die gleichen Funktionen darauf anwenden, die Sie vorher auf die Filmkritiken angewandt haben. Um einen Benutzer zufällig auszuwählen und andere Benutzer zu finden, die ähnliche Vorlieben wie er haben, geben Sie diesen Code in Ihrer Python-Session ein:

```
>> import random
>> user=delusers.keys()[random.randint(0,len(delusers)-1)]
>> user
u'veza'
>> recommendations.topMatches(delusers,user)
[(0.083, u'kuzz99'), (0.083, u'arturoochoa'), (0.083, u'NickSmith'), (0.083,
u'MichaelDahl'), (0.050, u'zinggoat')]
```

Sie können auch Link-Empfehlungen für diesen Benutzer erhalten, indem Sie `getRecommendations` aufrufen. Damit werden alle Elemente geordnet zurückgegeben, daher beschränkt man die Ausgabe besser auf die Top Ten:

```
>> recommendations.getRecommendations(delusers,user)[0:10]
[(0.278, u'http://www.devlisting.com/'),
(0.276, u'http://www.howtoforge.com/linux_ldap_authentication'),
(0.191, u'http://yarivsblog.com/articles/2006/08/09/secret-weapons-for-startups'),
(0.191, u'http://www.dadgum.com/james/performance.html'),
(0.191, u'http://www.codinghorror.com/blog/archives/000666.html')]
```

Natürlich kann diese Liste, wie schon zuvor gezeigt, transponiert werden, wodurch Sie Ihre Suchen auf Links statt auf Personen beziehen können. Um eine Reihe von Links zu finden, die einem ähneln, der Sie interessiert, können Sie dies versuchen:

e-bol.net

```
>> url=recommendations.getRecommendations(delusers,user)[0][1]
>> recommendations.topMatches(recommendations.transformPrefs(delusers),url)
[(0.312, u'http://www.fonttester.com/'),
(0.312, u'http://www.cssremix.com/'),
(0.266, u'http://www.logoorange.com/color/color-codes-chart.php'),
(0.254, u'http://yotophoto.com/'),
(0.254, u'http://www.wpdfd.com/editorial/basics/index.html')]
```

Das ist alles! Sie haben *del.icio.us* erfolgreich eine Empfehlungs-Engine hinzugefügt. Man kann hier natürlich noch eine Menge mehr tun. Da *del.icio.us* eine Suche nach Tags unterstützt, können Sie nach Tags Ausschau halten, die einander ähnlich sind. Sie können sogar nach Leuten fahnden, die versuchen, die »Popular«-Seiten zu manipulieren, indem sie die gleichen Links mit mehreren Konten posten.

Elementbasiertes Filtern

So, wie die Empfehlungs-Engine bisher implementiert ist, müssen alle Bewertungen von jedem Benutzer einfließen, um den Datenbestand aufzubauen. Das funktioniert vermutlich recht gut, wenn es ein paar Tausend Benutzer oder Objekte gibt, aber sehr große Sites wie Amazon haben Millionen von Kunden und Produkten – das Vergleichen eines Benutzers mit jedem anderen Benutzer und dann jedes Produkts, das die Benutzer bewertet haben, kann sehr langsam sein. Zudem wird eine Site, die Millionen von Produkten verkauft, zwischen den Personen nur wenige Überlappungen haben, was zu Problemen führen kann, wenn man entscheiden will, welche Personen ähnlich sind.

Die bisher genutzte Technik heißt *benutzerbasiertes kollaboratives Filtern*. Eine Alternative ist als *elementbasiertes kollaboratives Filtern* bekannt. In Fällen mit sehr großen Datenbeständen kann das elementbasierte kollaborative Filtern bessere Ergebnisse liefern, und es erlaubt zudem, dass viele der Berechnungen im Voraus ausgeführt werden können, sodass ein Benutzer, der Empfehlungen haben möchte, diese schneller erhalten kann.

Das Vorgehen bei elementbasiertem Filtern baut stark auf dem auf, was wir schon besprochen haben. Generell geht es darum, die ähnlichsten Elemente für jedes Element im Voraus zu berechnen. Wenn Sie dann einem Benutzer Empfehlungen geben wollen, schauen Sie sich die von ihm am besten gewerteten Elemente an und erstellen eine gewichtete Liste der Elemente, die diesen am ähnlichsten sind. Der entscheidende Unterschied liegt hier darin, dass Sie zwar im ersten Schritt alle Ihre Daten durchgehen müssen, sich die *Vergleiche zwischen Elementen aber nicht so oft ändern werden wie Vergleiche zwischen Benutzern*. Das bedeutet, dass Sie nicht fortlaufend die ähnlichsten Elemente zu jedem Element ermitteln müssen – Sie können das machen, wenn auf den Rechnern weniger los ist, oder auf einem Computer, der von Ihrer Hauptanwendung getrennt läuft.

e-bol.net

Die Item-Similarity-Daten erstellen

Um Elemente zu vergleichen, müssen Sie als Erstes eine Funktion schreiben, um den kompletten Datenbestand der ähnlichen Elemente aufzubauen. Wie gesagt – das muss nicht jedes Mal gemacht werden, wenn eine Empfehlung gebraucht wird, stattdessen bauen Sie den Datenbestand einmalig auf und verwenden ihn jedes Mal, wenn Sie ihn brauchen.

Um den Datenbestand zu erzeugen, fügen Sie die folgende Funktion der Datei *recommendations.py* hinzu:

```
def calculateSimilarItems(prefs,n=10):
  # Erstellen eines Dictionary mit Elementen, das zeigt, welche anderen
  # Elemente ihnen am ähnlichsten sind.
  result={}

  # Invertiert die Preference-Matrix, damit die Elemente im Fokus stehen.
  itemPrefs=transformPrefs(prefs)
  c=0
  for item in itemPrefs:
    # Statusaktualisierung für große Datenmengen
    c+=1
    if c%100==0: print "%d / %d" % (c,len(itemPrefs))
    # Die zu diesem Element ähnlichsten Elemente finden.
    scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
    result[item]=scores
  return result
```

Diese Funktion invertiert zuerst das Bewertungs-Dictionary mit der Funktion `transformPrefs`, die weiter oben definiert wurde, wobei eine Liste mit Elementen übergeben wird, in der steht, wie diese von jedem Benutzer bewertet wurden. Dann läuft die Funktion über jedes Element und übergibt jedes transformierte Dictionary an die Funktion `topMatches`, um die ähnlichsten Elemente zusammen mit deren Similarity Score zu erhalten. Schließlich erstellt die Funktion ein Dictionary mit Elementen zusammen mit einer Liste der ähnlichsten Elemente und gibt dieses zurück.

In Ihrer Python-Session bauen Sie die Element-Vergleichsdaten auf und schauen sich an, wie sie aussehen:

```
>>> reload(recommendations)
>>> itemsim=recommendations.calculateSimilarItems(recommendations.critics)
>>> itemsim
{'Lady in the Water': [(0.40000000000000002, 'You, Me and Dupree'),
                       (0.2857142857142857, 'The Night Listener'),...
 'Snakes on a Plane': [(0.22222222222222221, 'Lady in the Water'),
                       (0.18181818181818182, 'The Night Listener'),...
etc.
```

Denken Sie daran, dass diese Funktion nur so häufig laufen muss, dass die Elementähnlichkeiten halbwegs aktuell bleiben. Sie werden das häufiger tun müssen, solange die Anzahl der Benutzer noch klein und die Menge der Bewertungen gering

e-bol.net

ist, aber wenn es mehr Benutzer werden, werden auch die Similarity Scores zwischen den Elemente im Allgemeinen stabiler.

Empfehlungen erhalten

Jetzt sind Sie bereit, Empfehlungen mithilfe des Item-Similarity-Dictionary zu geben, ohne den gesamten Datenbestand durchlaufen zu müssen. Sie nehmen dafür alle Elemente, die der Benutzer bewertet hat, finden die ähnlichen Elemente und gewichten sie in Abhängigkeit von ihrer Ähnlichkeit. Um die Ähnlichkeiten zu finden, kann man einfach das Item-Dictionary nutzen.

Tabelle 2-3 zeigt den Prozess beim Finden der Empfehlungen für den elementbasierten Ansatz. Anders als in Tabelle 2-2 sind die Kritiker gar nicht mit von der Partie, stattdessen gibt es ein Raster mit Filmen, die ich bewertet habe, versus den Filmen, die ich nicht bewertet habe.

Tabelle 2-3: Elementbasierte Empfehlungen für Toby

Film	Bewertung	Night	B.xNight	Lady	B.xLady	Luck	B.xLuck
Snakes	4,5	0,182	0,818	0,222	0,999	0,105	0,474
Superman	4,0	0,103	0,412	0,091	0,363	0,065	0,258
Dupree	1,0	0,148	0,148	0,4	0,4	0,182	0,182
Gesamt		0,433	1,378	0,713	1,764	0,352	0,914
Normalisiert			3,183		2,598		2,473

Jede Zeile enthält einen Film, den ich schon gesehen habe, zusammen meiner persönlichen Bewertung. Für jeden Film, den ich noch nicht gesehen habe, gibt es eine Spalte, die angibt, wie ähnlich er den von mir bereits angeschauten Filmen ist – so hat zum Beispiel der Similarity Score zwischen *Superman* und *The Night Listener* den Wert 0,103. Die Spalten, die mit »B.x« beginnen, zeigen meine Bewertung des Films multipliziert mit der Ähnlichkeit – da ich *Superman* mit 4,0 bewertet habe, ist der Wert in der Spalte »Night« für Superman $4{,}0 \times 0{,}103 = 0{,}412$.

Die Zeile »Gesamt« zeigt die Summe der Similarity Scores und die Summe der B.x-Spalten für jeden Film. Um vorherzusagen, wie meine Bewertung für jeden Film sein könnte, teilen Sie einfach die Summe für die B.x-Spalte durch die Summe der Ähnlichkeitsspalte. Meine vorhergesagte Bewertung für *The Night Listener* ist daher $1{,}378/0{,}433 = 3{,}183$.

Sie können diese Funktionalität nutzen, indem Sie der Datei *recommendations.py* eine letzte Funktion hinzufügen:

```
def getRecommendedItems(prefs,itemMatch,user):
  userRatings=prefs[user]
  scores={}
  totalSim={}
```

e-bol.net

```
    # Über von diesem Benutzer bewertete Elemente laufen.
    for (item,rating) in userRatings.items( ):

      # Über Elemente laufen, die zu diesem ähnlich sind.
      for (similarity,item2) in itemMatch[item]:

        # Wenn dieser Benutzer schon bewertet hat, ignorieren.
        if item2 in userRatings: continue

        # Gewichtete Summe der Bewertung mal der Ähnlichkeit.
        scores.setdefault(item2,0)
        scores[item2]+=similarity*rating

        # Summe aller Ähnlichkeiten.
        totalSim.setdefault(item2,0)
        totalSim[item2]+=similarity

    # Jede Summe durch die aufsummierten Gewichtungen für den Durchschnitt teilen.
    rankings=[(score/totalSim[item],item) for item,score in scores.items( )]

    # Die Bewertungen sortiert zurückgeben.
    rankings.sort( )
    rankings.reverse( )
    return rankings
```

Sie können diese Funktion mit dem Ähnlichkeitsdatenbestand ausprobieren, den Sie weiter oben aufgebaut haben, um neue Empfehlungen für Toby zu erhalten:

```
>> reload(recommendations)
>> recommendations.getRecommendedItems(recommendations.critics,itemsim,'Toby')
[(3.182, 'The Night Listener'),
 (2.598, 'Just My Luck'),
 (2.473, 'Lady in the Water')]
```

The Night Listener liegt weiterhin mit deutlichem Vorsprung an erster Stelle, und *Just My Luck* hat mit *Lady in the Water* den Platz getauscht, auch wenn sie immer noch nahe beieinanderliegen. Wichtiger ist, dass beim Aufruf von `getRecommendedItems` nicht die gesamten Similarity Scores für alle anderen Kritiker berechnet werden müssen, weil der Item Similarity-Datenbestand im Voraus berechnet wurde.

Verwenden der MovieLens-Daten

Als letztes Beispiel wollen wir uns echte Filmbewertungen bei *MovieLens* anschauen. MovieLens wurde vom GroupLens-Projekt der University of Minnesota entwickelt. Sie können den Datenbestand unter *http://www.grouplens.org/node/73* herunterladen. Dabei gibt es zwei Datenbestände. Laden Sie die 100.000er-Daten abhängig von Ihrer Plattform entweder im Format *tar.gz* oder *zip* herunter.

e-bol.net

Das Archiv enthält eine Reihe von Dateien, aber die für uns interessanten sind *u.item*, in der sich eine Liste von Film-IDs und Titeln befindet, und *u.data*, die die eigentlichen Bewertungen in diesem Format enthält:

```
196  242   3      881250949
186  302   3      891717742
22   377   1      878887116
244  51    2      880606923
166  346   1      886397596
298  474   4      884182806
```

Jede Zeile hat eine Benutzer-ID, eine Film-ID, die Bewertung dieses Films durch diesen Benutzer und einen Zeitstempel. Sie können die Filmtitel ermitteln, aber die Benutzerdaten sind anonym, daher werden Sie in diesem Abschnitt nur mit Benutzer-IDs arbeiten. Die Daten enthalten Bewertungen von 1.682 Filmen durch 943 Benutzer, wobei jeder der Benutzer mindestens 20 Filme bewertet hat.

Erstellen Sie eine neue Methode `loadMovieLens` in *recommendations.py*, um die Daten zu laden:

```
def loadMovieLens(path='/data/movielens'):

  # Filmtitel laden.
  movies={}
  for line in open(path+'/u.item'):
    (id,title)=line.split('|')[0:2]
    movies[id]=title

  # Daten laden.
  prefs={}
  for line in open(path+'/u.data'):
    (user,movieid,rating,ts)=line.split('\t')
    prefs.setdefault(user,{})
    prefs[user][movies[movieid]]=float(rating)
  return prefs
```

In Ihrer Python-Session laden Sie nun die Daten; schauen Sie sich dann ein paar Bewertungen für irgendeinen Benutzer an:

```
>>> reload(recommendations)
>>> prefs=recommendations.loadMovieLens()
>>> prefs['87']
{'Birdcage, The (1996)': 4.0, 'E.T. the Extra-Terrestrial (1982)': 3.0,
 'Bananas (1971)': 5.0, 'Sting, The (1973)': 5.0, 'Bad Boys (1995)': 4.0,
 'In the Line of Fire (1993)': 5.0, 'Star Trek: The Wrath of Khan (1982)': 5.0,
'Speechless (1994)': 4.0, etc...
```

Jetzt können Sie benutzerbasierte Empfehlungen bekommen:

```
>>> recommendations.getRecommendations(prefs,'87')[0:30]
[(5.0, 'They Made Me a Criminal (1939)'), (5.0, 'Star Kid (1997)'),
 (5.0, 'Santa with Muscles (1996)'), (5.0, 'Saint of Fort Washington (1993)'),
 etc...]
```

e-bol.net

Abhängig von der Geschwindigkeit Ihres Computers kann es eine Weile dauern, wenn Sie die Empfehlungen auf diese Art und Weise erhalten. Das liegt daran, dass Sie jetzt mit einem deutlich größeren Datenbestand arbeiten. Je mehr Benutzer Sie haben, desto länger dauern benutzerbasierte Empfehlungen. Versuchen Sie es also stattdessen über die elementbasierten Empfehlungen:

```
>>> itemsim=recommendations.calculateSimilarItems(prefs,n=50)
100 / 1664
200 / 1664
etc...
>>> recommendations.getRecommendedItems(prefs,itemsim,'87')[0:30]
[(5.0, "What's Eating Gilbert Grape (1993)"), (5.0, 'Vertigo (1958)'),
 (5.0, 'Usual Suspects, The (1995)'), (5.0, 'Toy Story (1995)'),etc...]
```

Auch wenn das Aufbauen des Item Similarity-Dictionary eine ziemlich lange Zeit in Anspruch nimmt, sind die Empfehlungen dafür nach dem Aufbau fast sofort da. Zudem wird diese Zeit nicht mit zunehmenden Benutzern wachsen.

Das ist ein toller Datenbestand für Experimente, um zu sehen, wie verschiedene Vergleichsmethoden das Ergebnis beeinflussen, und um zu beobachten, wie performant element- bzw. benutzerbasierte Filtermethoden sind. Die Website von GroupLens bietet noch ein paar andere Datenbestände an, mit denen man spielen kann, unter anderem Bücher, Witze und noch mehr Filme.

Benutzerbasiertes oder elementbasiertes Filtern?

Elementbasiertes Filtern ist signifikant schneller als benutzerbasiertes Filtern, wenn man eine Liste mit Empfehlungen aus einem großen Datenbestand erhalten will, man erkauft sich diesen Vorteil aber durch zusätzlichen Aufwand beim Verwalten der Item Similarity Table. Zudem gibt es einen Unterschied in der Genauigkeit, der davon abhängt, wie »dünn« der Datenbestand ist. In dem Beispiel mit den Filmen ist der Datenbestand »dicht«, weil jeder Kritiker so gut wie jeden Film bewertet hat. Andererseits ist es recht unwahrscheinlich, zwei Personen zu finden, die die gleichen Lesezeichen bei *del.icio.us* gepflegt haben – die meisten Lesezeichen werden von einer kleinen Gruppe von Personen abgelegt, was zu einem dünn besetzten Datenbestand führt. Elementbasiertes Filtern ist normalerweise in dünn besetzten Datenbeständen schneller als benutzerbasiertes Filtern, während beide in dicht besetzten Daten ungefähr gleich schnell sind.

Um mehr über die Unterschiede bei der Performance dieser Algorithmen zu erfahren, lesen Sie den Artikel »Item-based Collaborative Filtering Recommendation Algorithms« von Sarwar et al. unter *http://citeseer.ist.psu.edu/sarwar01itembased.html*.

Andererseits ist das benutzerbasierte Filtern einfacher zu implementieren und spart einem auch die zusätzlichen Schritte, daher ist es meist die bessere Wahl, wenn es

e-bol.net

um kleinere Datenbestände geht, die in den Arbeitsspeicher passen und sich häufig ändern. Schließlich kann es in manchen Anwendungen durchaus von Nutzen sein, den Leuten zu zeigen, welche anderen Personen ähnliche Vorlieben haben wie sie selbst – vielleicht nicht auf einer Shopping-Site, aber eventuell auf einer Linksharing-Site oder bei Musikempfehlungen.

Sie haben nun gelernt, wie Sie Similarity Scores berechnen und wie Sie diese nutzen können, um Personen und Elemente zu vergleichen. Dieses Kapitel behandelte zwei verschiedene Empfehlungsalgorithmen – benutzerbasiert und elementbasiert. Dazu haben Sie mögliche Wege, die Vorlieben von Personen abzulegen, kennengelernt, sowie die API von *del.icio.us*, um ein Empfehlungssystem für Links aufzubauen. In Kapitel 9 werden Sie sehen, wie man, aufbauend auf einigen Ideen dieses Kapitels, Gruppen ähnlicher Personen mithilfe von unüberwachten Clustering-Algorithmen findet. Kapitel 9 wird noch einen alternativen Weg zum Zusammenbringen von Personen vorstellen, der dann nützlich ist, wenn Sie schon wissen, was für Leute diese Personen mögen.

Übungen

1. *Tanimoto Score*. Finden Sie heraus, was der Tanimoto Similarity Score ist. In welchen Fällen kann dieser Score als Ähnlichkeitsmetrik anstatt des euklidischen Abstands oder des Pearson-Korrelationskoeffizienten genutzt werden? Erstellen Sie eine neue Ähnlichkeitsfunktion, die den Tanimoto Score nutzt.
2. *Tag-Ähnlichkeit*. Erstellen Sie eine Datenmenge mit Tags und Elementen mithilfe der *del.icio.us*-API. Nutzen Sie diese, um die Ähnlichkeit zwischen Tags zu bestimmen, und versuchen Sie herauszufinden, ob es nahezu identische gibt. Finden Sie ein paar Elemente, die mit »programming« getaggt werden könnten, es aber nicht sind.
3. *Benutzerbasierte Effizienz*. Der benutzerbasierte Filteralgorithmus ist ineffizient, weil er einen Benutzer jedes Mal mit allen anderen Benutzern vergleicht, wenn eine Empfehlung notwendig ist. Schreiben Sie eine Funktion, die die Benutzerähnlichkeiten im Voraus berechnet, und passen Sie den Empfehlungscode so an, dass er nur die fünf ähnlichsten Benutzer beim Geben von Empfehlungen nutzt.
4. *Elementbasiertes Lesezeichen-Filtern*. Laden Sie eine Datenuntermenge von *del.icio.us* herunter und fügen Sie sie der Datenbank hinzu. Erstellen Sie eine Element-Element-Tabelle und verwenden Sie diese, um elementbasierte Empfehlungen für verschiedene Benutzer zu geben. Was für Unterschiede gibt es zu den benutzerbasierten Empfehlungen?
5. *Audioscrobbler*. Schauen Sie sich *http://www.audioscrobbler.net* an, ein Datenbestand, der Musikvorlieben für eine große Gruppe von Benutzern enthält. Nutzen Sie die API des entsprechenden Webservice, um Daten für ein Musik-Empfehlungssystem zu holen und dieses aufzubauen.

e-bol.net

KAPITEL 3

Gruppen bilden

In Kapitel 2 habe ich Wege vorgestellt, Objekte zu finden, die eng miteinander verbunden sind – so konnten Sie zum Beispiel jemanden finden, der in Bezug auf Filme einen ähnlichen Geschmack hat wie Sie. Dieses Kapitel baut darauf auf und führt in die *Clusteranalyse* ein, eine Methode, um Gruppen von Dingen, Personen oder Ideen zu bilden und zu visualisieren, die eng zusammengehören. In diesem Kapitel werden Sie erfahren, wie man Daten aus einer Vielzahl von Quellen aufbereitet, zwei verschiedene Clustering-Algorithmen kennenlernen, mehr zu Abstandsmetriken, einfachen Code zur grafischen Visualisierung der generierten Gruppen präsentiert bekommen und schließlich eine Methode kennenlernen, um sehr komplizierte Datensätze in zwei Dimensionen zu projezieren.

Die Clusteranalyse wird häufig in Anwendungen genutzt, die umfangreiche Datenmengen handhaben müssen. Händler, die die Kundenbestellungen protokollieren, können mit diesen Informationen automatisch Kundengruppen ermitteln, die ähnliche Kaufmuster aufweisen, und das neben den normalen demografischen Informationen. Leute gleichen Alters und mit gleichem Einkommen können trotzdem stark unterschiedliche Modevorstellungen haben, aber durch die Clusteranalyse lassen sich »Modeinseln« entdecken und dazu nutzen, eine Verkaufs- oder Marketing-Strategie zu entwickeln. Die Clusteranalyse wird auch ausgiebig in der Computational Biology beziehungsweise Bioinformatik angewendet, um Gengruppen zu finden, die ähnliche Verhaltensweisen zeigen, was die Chance eröffnet, dass sie auf eine Behandlung auf die gleiche Weise reagieren oder Teil der gleichen biologischen Signalwege sind.

Da es in diesem Buch um kollektive Intelligenz geht, kommen die Beispiele in diesem Kapitel aus Bereichen, in denen viele Personen verschiedene Informationen beitragen. Das erste Beispiel wird sich mit Blogs, den dort behandelten Themen und insbesondere der Verwendung von bestimmten Wörtern befassen, um zu zeigen, dass Blogs nach ihren Texten und Wörter nach ihrer Verwendung gruppiert werden können. Das zweite Beispiel wird einen Blick auf eine Community-Site werfen, auf

e-bol.net

der Menschen Dinge auflisten können, die ihnen gehören, und Dinge, die sie gern hätten. Wir werden diese Informationen nutzen, um zu zeigen, wie die Wünsche von Personen in Clustern guppiert werden können.

Überwachtes versus unüberwachtes Lernen

Techniken, die Beispieleingaben und -ausgaben verwenden, um zu lernen, wie Voraussagen zu machen sind, werden als *überwachte Lernmethoden* bezeichnet. Wir werden in diesem Buch viele überwachte Lernmethoden begutachten, einschließlich neuronaler Netze, Entscheidungsbäumen, Support-Vektor-Maschinen und Bayes-Filtern. Anwendungen, die diese Methoden nutzen, »lernen«, indem sie eine Reihe von Eingaben und die gewünschten Ausgaben analysieren. Wenn wir Informationen mithilfe einer dieser Methoden extrahieren wollen, geben wir Eingabewerte an und erwarten von der Anwendung, dass sie eine Ausgabe auf Grundlage dessen liefert, was sie bisher gelernt hat.

Die Clusteranalyse ist ein Beispiel für *unüberwachtes Lernen*. Anders als ein neuronales Netz oder ein Entscheidungsbaum werden unüberwachte Lernalgorithmen nicht mit Beispielen für richtige Antworten trainiert. Ihr Zweck liegt darin, Strukturen in einem Datenbestand zu finden, bei denen keine direkte Antwort das Ziel ist. Im Modebeispiel weiter oben teilen die Cluster dem Händler nicht mit, was eine einzelne Person vermutlich kaufen wird, und sie machen auch keine Vorhersagen darüber, zu welcher Modeinsel eine neue Person passen wird. Das Ziel der Clusteranalyse ist, aus den gegebenen Daten die darin bestehenden unterschiedlichen Gruppen zu ermitteln. Zu anderen Beispielen für unüberwachtes Lernen gehören die *Nicht-negative Matrix-Faktorisierung*, die wir in Kapitel 10 behandeln werden, und *selbstorganisierende Karten*.

Wortvektoren

Normalerweise bereitet man die Daten zur Clusteranalyse auf, indem man eine gemeinsame Menge numerischer Attribute bestimmt, die zum Vergleichen der Elemente genutzt werden kann. Das ähnelt sehr dem Vorgehen in Kapitel 2, bei dem die Bewertungen der Kritiker für die Menge der gemeinsam bewerteten Filme verglichen wurden und das Vorhandensein oder Nichtvorhandensein eines Lesezeichens für *del.icio.us*-Benutzer in eine 1 oder 0 übersetzt wurde.

Einordnen der Blogger

Dieses Kapitel wird ein paar Beispiel-Datenbestände nutzen. In der ersten Datenmenge werden die zu gruppierenden Elemente 120 Top-Blogs sein und die Daten, aufgrund deren sie geclustert werden sollen, das Vorkommen bestimmter Wörter in den Blog-Feeds. Eine kleine Untermenge davon ist in Tabelle 3-1 zu sehen.

e-bol.net

Tabelle 3-1: Untermenge der Wortfrequenz in den Blogs

	»china«	»kids«	»music«	»yahoo«
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

Indem die Blogs basierend auf der Wortfrequenz geclustert werden, kann man möglicherweise erkennen, dass es Gruppen von Blogs gibt, die regelmäßig ähnliche Themen behandeln oder einen ähnlichen Stil pflegen. Solch ein Ergebnis kann sehr nützlich sein, wenn man die große Anzahl an Blogs durchsuchen, katalogisieren und durchstöbern will, die momentan online sind.

Um diesen Datenbestand zu erzeugen, müssen Sie die Feeds für eine Reihe von Blogs herunterladen, den Text aus den Einträgen extrahieren und eine Tabelle mit Wortfrequenzen aufbauen. Wenn Sie diese Schritte gern überspringen würden, können Sie die Daten von *http://kiwitobes.com/clusters/blogdata.txt* herunterladen.

Die Wörter in einem Feed zählen

So gut wie alle Blogs können online oder über ihre *RSS-Feeds* gelesen werden. Ein RSS-Feed ist ein einfaches XML-Dokument, das Informationen über das Blog und alle seine Einträge enthält. Beim Erzeugen der Wortfrequenz für jedes Blog muss man als Erstes diese Feeds parsen. Glücklicherweise gibt es dafür schon ein ausgezeichnetes Modul namens Universal Feed Parser, das Sie unter *http://www.feedparser.org* herunterladen können.

Dieses Modul erleichtert es, die Titel, Links und Einträge eines beliebigen RSS- oder Atom-Feeds zu erhalten. Als Nächstes müssen Sie eine Funktion erstellen, die alle Wörter aus einem Feed extrahiert. Erstellen Sie eine neue Datei namens *generatefeedvector.py* und fügen Sie den folgenden Code ein:

```
import feedparser
import re

# Liefert den Titel und ein Dictionary mit Wortfrequenzen für einen RSS-Feed.
def getwordcounts(url):
  # Feed parsen.
  d=feedparser.parse(url)
  wc={}

  # Über alle Einträge laufen lassen.
  for e in d.entries:
    if 'summary' in e: summary=e.summary
    else: summary=e.description

    # Wortliste erstellen.
    words=getwords(e.title+' '+summary)
```

e-bol.net

```
    for word in words:
      wc.setdefault(word,0)
      wc[word]+=1
  return d.feed.title,wc
```

RSS- und Atom-Feeds haben immer einen Titel und eine Liste mit Einträgen. Jeder Eintrag enthält normalerweise entweder das Tag `summary` oder `description`, in dem der eigentlichen Text des Eintrags steht. Die Funktion `getwordcounts` übergibt diese Zusammenfassung an `getwords`, die das ganze HTML löscht und die Wörter an den Zeichen aufteilt, die keine Buchstaben sind. Schließlich gibt sie dies als Liste zurück. Fügen Sie die Funktion `getwords` der Datei *generatefeedvector.py* hinzu:

```
def getwords(html):
  # Alle HTML-Tags entfernen.
  txt=re.compile(r'<[^>]+>').sub('',html)

  # Wörter an Nicht-Buchstaben aufteilen.
  words=re.compile(r'[^A-Z^a-z]+').split(txt)

  # In Kleinbuchstaben umwandeln.
  return [word.lower( ) for word in words if word!='']
```

Nun brauchen Sie eine Liste von Feeds, mit denen Sie arbeiten können. Wenn Sie wollen, können Sie sich solch eine Liste mit Feed-URLs selbst zusammenstellen, oder aber Sie nutzen eine vorgefertigte Liste mit 100 RSS-URLs. Diese Liste wurde erstellt, indem die Feeds für die am häufigsten referenzierten (englischsprachigen) Blogs gesammelt und die Einträge entfernt wurden, die nicht die vollständigen Texte ihrer Einträge oder hauptsächlich Bilder enthielten. Sie können die Liste unter *http://kiwitobes.com/clusters/feedlist.txt* herunterladen.

Dabei handelt es sich um eine reine Textdatei mit einer URL in jeder Zeile. Wenn Sie Ihr eigenes Blog oder bestimmte Favoriten haben und sehen möchten, wie sie sich im Vergleich zu einigen der beliebtesten Blogs machen, können Sie diese der Datei hinzufügen.

Der Code für das Durchlaufen der Feeds und das Generieren der Daten wird der Hauptcode in *generatefeedvector.py* sein (also alles, was nicht in einer Funktion steckt). Der erste Teil des Codes durchläuft jede Zeile in *feedlist.txt* und generiert die Wortfrequenz für jedes Blog sowie die Anzahl an Blogs, in denen ein Wort auftaucht (`apcount`). Fügen Sie diesen Code an das Ende von *generatefeedvector.py* an:

```
apcount={}
wordcounts={}
for feedurl in file('feedlist.txt'):
  title,wc=getwordcounts(feedurl)
  wordcounts[title]=wc
  for word,count in wc.items( ):
    apcount.setdefault(word,0)
    if count>1:
      apcount[word]+=1
```

e-bol.net

Als Nächstes generieren wir die Liste mit den Wörtern, die tatsächlich beim Zählen in jedem Blog genutzt werden. Da Wörter wie »the« nahezu in jedem auftauchen werden und andere wie »flim-flam« vielleicht nur in einem, können Sie die Gesamtzahl an Testwörtern reduzieren, indem Sie nur diejenigen wählen, die sich zwischen der maximalen und minimalen Prozentzahl bewegen. In diesem Fall beginnen Sie mit 10% als Untergrenze und 50% als Obergrenze, aber es lohnt sich, mit diesen Zahlen zu experimentieren, wenn zu viele Standard- oder seltsame Wörter auftauchen:

```
wordlist=[]
for w,bc in apcount.items():
  frac=float(bc)/len(feedlist)
  if frac>0.1 and frac<0.5: wordlist.append(w)
```

Schließlich nutzen Sie die Wortliste und die Blog-Liste, um eine Textdatei mit einer großen Matrix zu erzeugen, die alle Wortfrequenzen für jedes Blog enthält:

```
out=file('blogdata.txt','w')
out.write('Blog')
for word in wordlist: out.write('\t%s' % word)
out.write('\n')
for blog,wc in wordcounts.items():
  out.write(blog)
  for word in wordlist:
    if word in wc: out.write('\t%d' % wc[word])
    else: out.write('\t0')
  out.write('\n')
```

Um die Wortfrequenzdatei zu erstellen, rufen Sie *generatefeedvector.py* an der Befehlszeile auf:

```
c:\code\blogcluster>python generatefeedvector.py
```

Das Herunterladen all dieser Feeds kann ein paar Minuten dauern, aber schließlich wird eine Datei mit dem Namen *blogdata.txt* erzeugt worden sein. Öffnen Sie diese Datei, um zu kontrollieren, ob sie eine durch Tabs getrennte Tabelle enthält, in deren Spalten die Wörter und in deren Zeilen die Blogs stehen. Dieses Dateiformat wird von den Funktionen in diesem Kapitel verwendet werden, sodass Sie später eine andere Datenmenge erstellen oder sogar ein entsprechend formatiertes Arbeitsblatt einer Tabellenkalkulation als per Tabs getrennte Textdatei speichern können, um diese mit den Clusteralgorithmen zu verwenden.

Hierarchische Clusteranalyse

Die hierarchische Clusteranalyse führt zu einer Hierarchie von Gruppen, indem sie immer wieder die zwei ähnlichsten Gruppen miteinander verschmilzt. Jede dieser Gruppen besteht zu Beginn aus einem einzelnen Element, in diesem Fall aus einem einzelnen Blog. Bei jedem Iterationsschritt berechnet diese Methode den Abstand zwischen jeweils zwei Gruppen und verbindet die zwei am dichtesten zusammenlie-

e-bol.net

genden zu einer neuen Gruppe. Dies wird wiederholt, bis es nur noch eine Gruppe gibt. Abbildung 3-1 zeigt diesen Prozess.

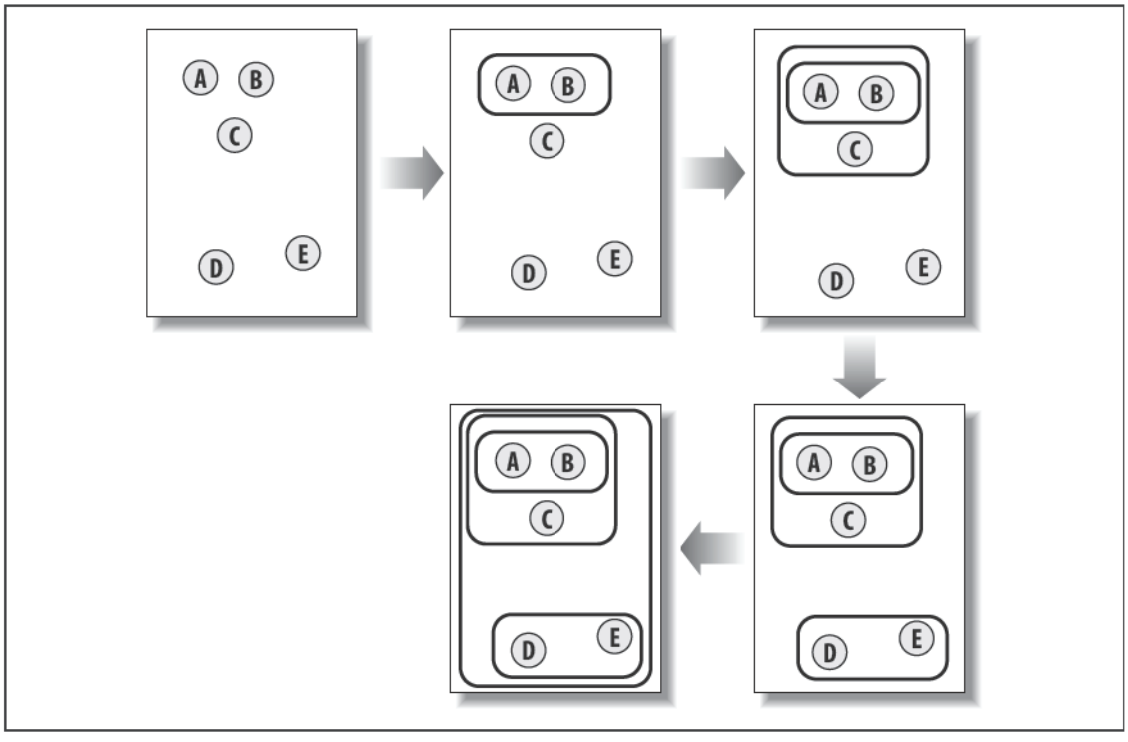

Abbildung 3-1: Ablauf der hierarchischen Clusteranalyse

In dieser Abbildung wird die Ähnlichkeit der Elemente durch ihre relative Position dargestellt – je näher sich zwei Elemente sind, desto ähnlicher sind sie auch. Zuerst bestehen die Gruppen einfach aus den einzelnen Elementen. Im zweiten Schritt können Sie sehen, dass A und B, die beiden am nahesten beieinanderliegenden Elementen, zu einer neuen Gruppe verbunden wurden, deren Position in der Mitte der beiden Elemente liegt. Im dritten Schritt wird diese neue Gruppe mit C verbunden. Da D und E nun die nächsten am engsten beieinanderliegenden Elemente sind, bilden diese eine neue Gruppe. Im letzten Schritt werden die noch verbleibenden zwei Gruppen verbunden.

Ist die hierarchische Clusteranalyse abgeschlossen, betrachten Sie die Ergebnisse normalerweise am besten in einem Graphen, der als *Dendrogramm* bezeichnet wird und die Knoten als Hierarchie angeordnet darstellt. Das Dendrogramm des obigen Beispiels ist in Abbildung 3-2 abgebildet.

Dieses Dendrogramm nutzt die Verbindungen nicht nur, um zu zeigen, welche Elemente schließlich in welchem Cluster gelandet sind, es nutzt auch noch den Abstand, um zu zeigen, wie weit die Elemente voneinander entfernt sind. Der Cluster AB liegt deutlich näher an den einzelnen Elementen A und B als der Cluster DE bei den Elementen D und E. Rendert man den Graphen auf diese Art und Weise,

e-bol.net

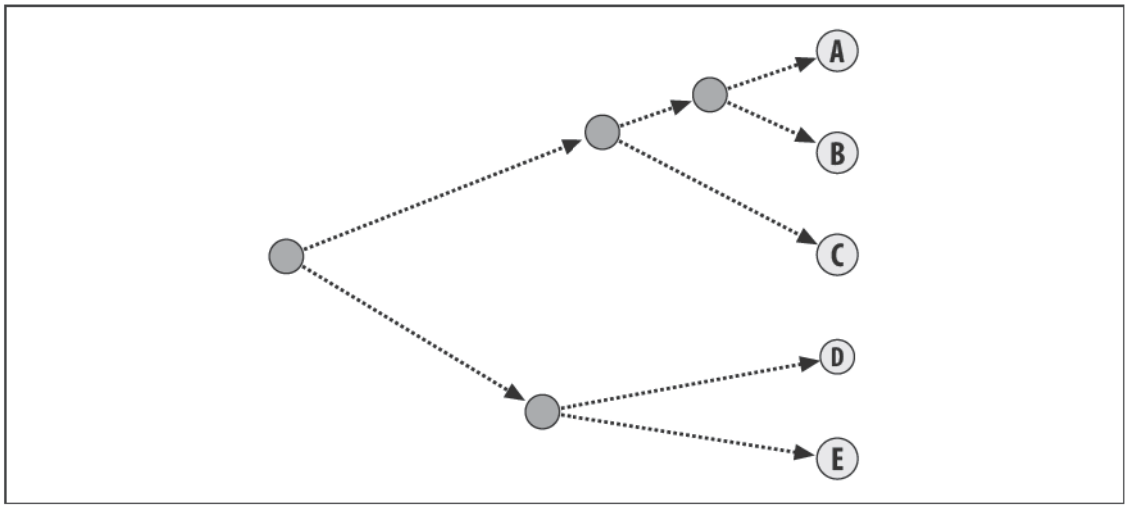

Abbildung 3-2: Ein Dendrogramm ist eine grafische Darstellung der hierarchischen Clusteranalyse

kann das dabei helfen zu bestimmen, wie ähnlich die Elemente in einem Cluster sind, was als *Engheit* (*Tightness*) des Clusters gedeutet werden kann.

Dieser Abschnitt wird Ihnen zeigen, wie Sie die Blog-Daten in Gruppen aufteilen können, um eine Hierarchie von Blogs zu erstellen, die bei Erfolg dazu führt, dass die Blogs thematisch gruppiert werden. Zunächst brauchen Sie eine Methode, um die Daten aus der Datei zu laden. Erstellen Sie eine Datei namens *clusters.py* und fügen Sie diese Funktion ein:

```
def readfile(filename):
  lines=[line for line in file(filename)]

  # Erste Zeile enthält die Spaltentitel.
  colnames=lines[0].strip( ).split('\t')[1:]
  rownames=[]
  data=[]
  for line in lines[1:]:
    p=line.strip( ).split('\t')
    # Erste Spalte in jeder Zeile ist der Zeilenname.
    rownames.append(p[0])
    # Die Daten dieser Zeile sind der Rest der Zeile.
    data.append([float(x) for x in p[1:]])
  return rownames,colnames,data
```

Diese Funktion liest die erste Zeile in die Liste mit den Spaltennamen und die erste Spalte in eine Liste mit Zeilennamen ein, dann steckt sie alle restlichen Daten in eine große Liste, in der jedes Element die Daten für diese Zeile enthält. Den Wert für eine Zelle kann man über seine Zeile und Spalte in `data` ansprechen, was auch den Indizes der Listen `rownames` und `colnames` entspricht.

Im nächsten Schritt definiert man die *Nähe*. Wir haben das schon in Kapitel 2 behandelt, als wir den euklidischen Abstand und den Pearson-Korrelationskoeffizi-

e-bol.net

enten als Beispiele beim Bestimmen der Ähnlichkeit von zwei Filmkritikern genutzt haben. Im hier vorliegenden Beispiel enthalten manche Blogs mehr oder deutlich längere Einträge als andere, was auch zu insgesamt mehr Wörtern führt. Der Pearson-Korrelationskoeffizient gleicht das aus, da er wirklich versucht zu bestimmen, wie gut zwei Datenmengen auf eine Gerade passen. Der Code für den Korrelationskoeffizienten für dieses Modul erwartet zwei Listen mit Zahlen und liefert ihren Korrelationskoeffizienten zurück:

```
from math import sqrt
def pearson(v1,v2):
  # einfache Summen
  sum1=sum(v1)
  sum2=sum(v2)

  # Summen der Quadrate
  sum1Sq=sum([pow(v,2) for v in v1])
  sum2Sq=sum([pow(v,2) for v in v2])

  # Summe der Produkte
  pSum=sum([v1[i]*v2[i] for i in range(len(v1))])

  # r (Pearson-Korrelationskoeffizent) berechnen
  num=pSum-(sum1*sum2/len(v1))
  den=sqrt((sum1Sq-pow(sum1,2)/len(v1))*(sum2Sq-pow(sum2,2)/len(v1)))
  if den==0: return 0

  return 1.0-num/den
```

Denken Sie daran, dass der Korrelationskoeffizient den Wert 1,0 hat, wenn zwei Elemente perfekt zusammenpassen, und nahe bei 0,0 liegt, wenn es überhaupt keine Beziehung zueinander gibt. Die letzte Codezeile liefert 1,0 minus dem Korrelationskoeffizienten zurück, um einen kleineren Abstand zwischen Elementen zurückzugeben, die sich ähnlicher sind.

Jeder Cluster in einem Algorithmus zur hierarchischen Clusteranalyse ist entweder ein Punkt im Baum mit zwei Zweigen oder ein Endpunkt, der mit einer Zeile in der Datenmenge (in diesem Fall einem Blog) verbunden ist. Jeder Cluster enthält zudem Daten über seine Position, die entweder für die Endpunkte die Zeilendaten oder bei den anderen Knotentypen die verbundenen Daten aus seinen zwei Zweigen ist. Sie können eine Klasse namens `bicluster` erstellen, die alle diese Eigenschaften enthält, um damit den hierarchischen Baum zu repräsentieren. Erstellen Sie den Clustertyp als Klasse in *cluster.py*:

```
class bicluster:
  def __init__(self,vec,left=None,right=None,distance=0.0,id=None):
    self.left=left
    self.right=right
    self.vec=vec
    self.id=id
    self.distance=distance
```

e-bol.net

Der Algorithmus für die hierarchische Clusteranalyse beginnt damit, eine Gruppe von Clustern zu erzeugen, die genau die ursprünglichen Elemente enthalten. Die Hauptschleife der Funktion sucht nach den am besten zusammenpassenden zwei Elementen, indem sie jedes mögliche Paar ausprobiert und ihre Korrelation berechnet. Das beste Clusterpaar wird dann in einen einzelnen Cluster verbunden. Die Daten für diesen neuen Cluster sind der Durchschnitt der Daten aus den beiden alten Clustern. Dieser Prozess wird wiederholt, bis nur noch ein Cluster übrig bleibt. Es kann sehr zeitaufwendig sein, all diese Berechnungen durchzuführen, daher sollten Sie die Korrelationsergebnisse für jedes Paar abspeichern, da sie wieder und wieder berechnet werden müssten, bis eines der Elemente im Paar in einen anderen Cluster eingebaut wird.

Fügen Sie den Algorithmus `hcluster` in *clusters.py* ein:

```
def hcluster(rows,distance=pearson):
  distances={}
  currentclustid=-1

  # Cluster sind zuerst einfach die Zeilen.
  clust=[bicluster(rows[i],id=i) for i in range(len(rows))]

  while len(clust)>1:
    lowestpair=(0,1)
    closest=distance(clust[0].vec,clust[1].vec)

    # Jedes Paar nach dem kleinsten Abstand durchlaufen.
    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        # distances ist der Cache für die Abstandsberechnungen.
        if (clust[i].id,clust[j].id) not in distances:
          distances[(clust[i].id,clust[j].id)]=distance(clust[i].vec,clust[j].vec)

        d=distances[(clust[i].id,clust[j].id)]

        if d<closest:
          closest=d
          lowestpair=(i,j)

    # Berechnen des Durchschnitts der beiden Cluster.
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # Einen neuen Cluster erstellen.
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # Cluster-IDs, die nicht aus den Ursprungsdaten kommen, sind negativ.
    currentclustid-=1
    del clust[lowestpair[1]]
```

e-bol.net

```
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]
```

Da jeder Cluster auf die zwei Cluster verweist, aus denen er zusammengebaut wurde, kann der von dieser Funktion zurückgegebene Cluster rekursiv durchsucht werden, um alle Cluster und ihre Endknoten erneut aufzubauen.

Zum Ausführen der hierarchischen Clusteranalyse öffnen Sie eine Python-Session, laden die Datei und rufen `hcluster` für die Daten auf:

```
$ python
>> import clusters
>> blognames,words,data=clusters.readfile('blogdata.txt')
>> clust=clusters.hcluster(data)
```

Das kann ein paar Minuten dauern. Das Puffern der Abstände verbessert die Geschwindigkeit deutlich, aber der Algorithmus muss immer noch die Korrelation zwischen jedem Paar von Blogs berechnen. Dieser Prozess kann beschleunigt werden, indem man eine externe Bibliothek zum Berechnen der Abstände nutzt. Um Ihre Ergebnisse anzuzeigen, können Sie eine einfache Funktion erstellen, die den Clusterbaum rekursiv durchläuft und ihn wie eine Dateisystemhierarchie ausgibt. Fügen Sie die Funktion `printclust` der Datei *clusters.py* hinzu:

```
def printclust(clust,labels=None,n=0):
  # Einrücken, um ein hierarchisches Layout zu erhalten.
  for i in range(n): print ' ',
  if clust.id<0:
    # Negative ID bedeutet, dass dies eine Verzweigung ist.
    print '-'
  else:
    # Positive ID bedeutet, dass dies ein Endpunkt ist.
    if labels==None: print clust.id
    else: print labels[clust.id]

  # Nun den rechten und linken Zweig ausgeben.
  if clust.left!=None: printclust(clust.left,labels=labels,n=n+1)
  if clust.right!=None: printclust(clust.right,labels=labels,n=n+1)
```

Diese Ausgabe sieht nicht sehr toll aus und lässt sich bei großen Datenbeständen wie der Blog-Liste nur schlecht lesen, aber sie gibt einen guten Überblick darüber, ob die Clusteranalyse funktioniert. Im nächsten Abschnitt werden wir eine grafische Version erstellen, die sich viel besser lesen lässt und maßstabsgerecht gezeichnet ist, um die Ausdehnung jedes Clusters zu zeigen.

Rufen Sie diese Funktion in Ihrer Python-Session für die gerade erzeugten Cluster auf:

```
>> reload(clusters)
>> clusters.printclust(clust,labels=blognames)
```

e-bol.net

Die Ausgabe wird alle 100 Blogs enthalten und daher recht lang sein. Hier ein Beispiel für einen Cluster, den ich in diesem Datenbestand gefunden habe:

```
John Battelle's Searchblog
-
  Search Engine Watch Blog
  -
    Read/WriteWeb
    -
      Official Google Blog
      -
        Search Engine Roundtable
        -
          Google Operating System
          Google Blogoscoped
```

Die ursprünglichen Elemente werden hier gezeigt. Die Striche stehen für einen Cluster aus zwei oder mehr verschmolzenen Elementen. Hier sehen Sie ein funktionierendes Beispiel für das Finden einer Gruppe – es ist auch interessant zu sehen, dass es solch einen großen Bereich von suchbezogenen Blogs unter den beliebtesten Feeds gibt. Wenn Sie sich die gesamten Daten anschauen, sollten Sie auch Cluster mit politischen Blogs, technologischen Blogs und Blogs über das Bloggen entdecken können.

Sie werden vermutlich auch ein paar Anomalien finden. Die Autoren haben vielleicht nicht über die gleichen Themen geschrieben, aber der Clusteralgorithmus sagt, dass die Wortfrequenzen korreliert sind. Das kann den Schreibstil widerspiegeln oder einfach ein Zufall sein, der von dem Tag abhängt, an dem die Daten heruntergeladen wurden.

Zeichnen des Dendrogramms

Sie können die Cluster besser interpretieren, wenn Sie sie als Dendrogramm zeichnen. Die Ergebnisse von hierarchischen Clusteranalysen werden normalerweise so dargestellt, denn Dendrogramme bringen eine Menge Informationen auf relativ kleinem Raum unter. Da Dendrogramme grafischer Natur sind und als JPGs gespeichert werden, müssen Sie die Python Imaging Library (PIL) herunterladen, die Sie unter *http://pythonware.com* erhalten.

Diese Bibliothek bringt einen Installer für Windows und Source-Distributionen für andere Plattformen mit. Weitere Informationen zum Herunterladen und Installieren der PIL erhalten Sie in Anhang A. Die PIL macht es sehr einfach, Bilder mit Text und Linien zu erzeugen – und dies reicht aus, um ein Dendrogramm zu erstellen. Fügen Sie am Anfang von *clusters.py* die Anweisung `import` hinzu:

```
from PIL import Image,ImageDraw
```

e-bol.net

Als Erstes brauchen Sie eine Funktion, die die Gesamthöhe eines gegebenen Clusters ermittelt. Wenn Sie die Gesamthöhe des Bilds und die Position der verschiedenen Knoten bestimmen wollen, müssen Sie deren einzelne Höhen kennen. Ist ein Cluster ein Endpunkt (also ohne Verzweigungen), ist dessen Höhe 1 und ansonsten die Summe der Höhen seiner Zweige. Das lässt sich leicht als rekursive Funktion definieren, die Sie *clusters.py* hinzufügen können:

```
def getheight(clust):
  # Ist dies ein Endpunkt? Dann ist die Höhe 1.
  if clust.left==None and clust.right==None: return 1

  # Ansonsten ist die Höhe die Summe der Höhen der Verzweigungen.
  return getheight(clust.left)+getheight(clust.right)
```

Was Sie noch wissen müssen, ist der Gesamtfehler des Wurzelknotens. Da die Länge der Linien danach bemessen wird, wie groß der Fehler in jedem Knoten ist, erzeugen Sie einen Skalierungsfaktor auf Basis des Gesamtfehlers. Die Fehlertiefe eines Knotens ist einfach der maximal mögliche Fehler jedes seiner Zweige:

```
def getdepth(clust):
  # Der Abstand eines Endpunkts ist 0.0.
  if clust.left==None and clust.right==None: return 0

  # Der Abstand eines Zweigs ist der größere seiner beiden Seiten
  # plus der eigene Abstand.
  return max(getdepth(clust.left),getdepth(clust.right))+clust.distance
```

Die Funktion `drawdendrogram` erzeugt ein neues Bild mit 20 Pixeln Höhe und einer festen Breite für jeden abschließenden Cluster. Der Skalierungsfaktor wird ermittelt, indem die feste Breite durch die totale Tiefe geteilt wird. Die Funktion erstellt ein Zeichenobjekt für dieses Bild und ruft dann `drawnode` für den Wurzelknoten auf, wobei sie der Funktion mitteilt, dass dessen Position vertikal in der Mitte auf der linken Seite des Bilds liegen soll.

```
def drawdendrogram(clust,labels,jpeg='clusters.jpg'):
  # Höhe und Breite
  h=getheight(clust)*20
  w=1200
  depth=getdepth(clust)

  # Breite ist fest, daher wird der Maßstab entsprechend angepasst.
  scaling=float(w-150)/depth

  # Erzeugen eines neuen Bilds mit einem weißen Hintergrund.
  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  draw.line((0,h/2,10,h/2),fill=(255,0,0))

  # Den ersten Knoten zeichnen.
  drawnode(draw,clust,10,(h/2),scaling,labels)
  img.save(jpeg,'JPEG')
```

e-bol.net

Die wichtige Funktion ist hier `drawnode`, die ein Cluster und seine Position erwartet. Sie nimmt die Höhe der Kindknoten, berechnet, wo sie liegen sollten, und zeichnet Linien zu ihnen – eine lange vertikale Linie und zwei horizontale Linien. Die Länge der horizontalen Linien hängt davon ab, wie groß der Fehler im Cluster ist. Längere Linien zeigen, dass die beiden Cluster, die zu dem einen Cluster verbunden wurden, nicht sehr ähnlich waren, während kürzere Linien darauf hinweisen, dass sie nahezu identisch sind. Fügen Sie die Funktion `drawnode` der Datei *clusters.py* hinzu:

```
def drawnode(draw,clust,x,y,scaling,labels):
  if clust.id<0:
    h1=getheight(clust.left)*20
    h2=getheight(clust.right)*20
    top=y-(h1+h2)/2
    bottom=y+(h1+h2)/2
    # Linienlänge
    ll=clust.distance*scaling
    # vertikale Linie von diesem Cluster zu den Kindern
    draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))

    # horizontale Linie zum linken Element
    draw.line((x,top+h1/2,x+ll,top+h1/2),fill=(255,0,0))

    # horizontale Linie zum rechten Element
    draw.line((x,bottom-h2/2,x+ll,bottom-h2/2),fill=(255,0,0))

    # Aufruf der Funktion, um den linken und den rechten Knoten zu zeichnen.
    drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
    drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
  else:
    # Wenn dies ein Endpunkt ist, Zeichnen des Elementnamens.
    draw.text((x+5,y-7),labels[clust.id],(0,0,0))
```

Um das Bild zu erzeugen, wechseln Sie zu Ihrer Python-Session und geben Folgendes ein:

```
>> reload(clusters)
>> clusters.drawdendrogram(clust,blognames,jpeg='blogclust.jpg')
```

Dies erzeugt eine Datei namens *blogclust.jpg* mit dem Dendrogramm. Das Dendrogramm sollte so ähnlich aussehen wie das in Abbildung 3-3. Wenn Sie mögen, können Sie die Höhe und Breite anpassen, um es besser ausdrucken zu können oder um es übersichtlicher zu gestalten.

Spalten-Clusteranalyse

Es ist häufig notwendig, sowohl die Zeilen als auch die Spalten einer Clusteranalyse zu unterziehen. In einer Marketing-Studie kann es interessant sein, Personen zu gruppieren, um demografische Informationen und Produkte zu finden oder auch um die Regalplatzierung von Produkten zu bestimmen, die häufig zusammen

e-bol.net

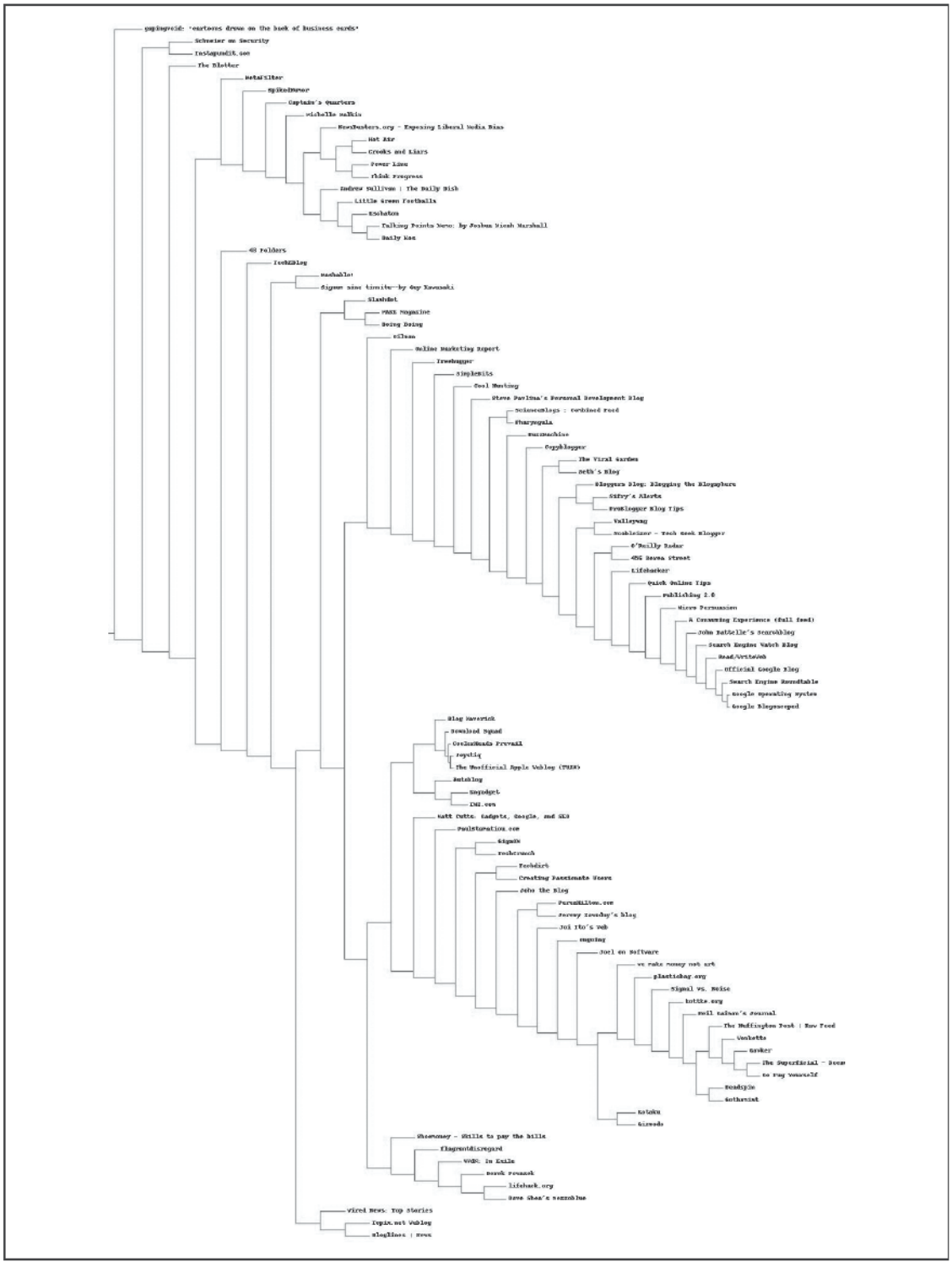

Abbildung 3-3: Dendrogramm mit Blog-Clustern

gekauft werden. In der Blog-Datenmenge repräsentieren die Spalten die Wörter, und es ist eventuell interessant zu sehen, welche Wörter häufig zusammen genutzt werden.

e-bol.net

Der einfachste Weg, dies mit den schon geschriebenen Funktionen zu erreichen, ist, den gesamten Datenbestand zu drehen, sodass die Spalten (die Wörter) zu Zeilen werden – jede mit einer Liste von Zahlen, die angibt, wie oft dieses Wort in jedem der Blogs auftaucht. Fügen Sie diese Funktion der Datei *clusters.py* hinzu:

```
def rotatematrix(data):
  newdata=[]
  for i in range(len(data[0])):
    newrow=[data[j][i] for j in range(len(data))]
    newdata.append(newrow)
  return newdata
```

Sie können jetzt die Matrix rotieren und die gleichen Operationen zur Clusteranalyse und zum Zeichnen des Dendrogramms aufrufen. Da es viel mehr Wörter als Blogs gibt, wird dies länger dauern als beim Blog-Clustern. Denken Sie daran, dass durch das Drehen der Matrix nun die Wörter und nicht mehr die Blogs als Bezeichner auftauchen.

```
>> reload(clusters)
>> rdata=clusters.rotatematrix(data)
>> wordclust=clusters.hcluster(rdata)
>> clusters.drawdendrogram(wordclust,labels=words,jpeg='wordclust.jpg')
```

Bei der Clusteranalyse muss man immer berücksichtigen, dass die Wahrscheinlichkeit für unsinnige Cluster bei deutlich mehr Elementen als Variablen zunimmt. Es gibt viel mehr Wörter als Blogs, daher werden Sie bei den Blog-Clustern mehr sinnvolle Muster als bei den Wort-Clustern erkennen. Trotzdem zeigen sich auch dort ein paar interessante Cluster, wie in Abbildung 3-4 zu sehen ist.

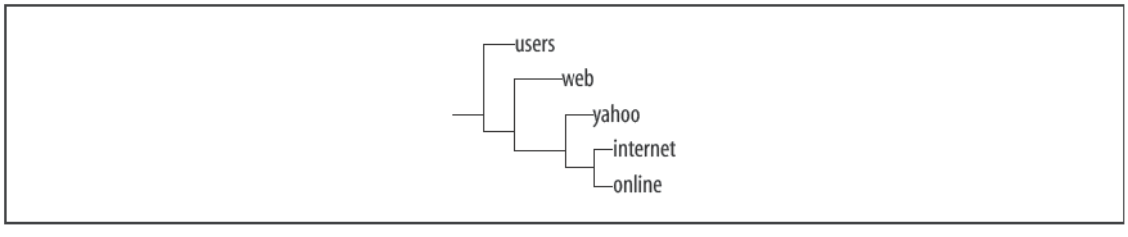

Abbildung 3-4: Wort-Cluster mit Onlineservice-bezogenen Wörtern

Die Cluster zeigen klar, dass eine Reihe von Wörtern in Blogs häufig zusammen auftaucht, die von Onlineservices oder internetbezogene Themen handeln. Es ist möglich, auch andere Cluster zu finden, die bestimmte Nutzungsmuster widerspiegeln, wie zum Beispiel »fact«, »us«, »say«, »very« und »think«, die zeigen, dass ein Blog ziemlich eigensinnig schreibt.

K-Means-Clusteranalyse

Hierarchische Clusteranalysen haben als Ergebnis einen hübschen Baum, aber auch ein paar Nachteile. Die Baumansicht teilt die Daten nicht direkt ohne zusätzliche

e-bol.net

Arbeit in einzelne Gruppen auf, und der Algorithmus ist extrem rechenaufwendig. Da die Beziehung zwischen jedem Paar von Elementen und auch bei jeder Verschmelzung von Elementen berechnen werden muss, ist der Algorithmus bei sehr großen Datenmengen langsam.

Eine alternative Clustermethode ist die *K-Means*-Clusteranalyse. Diese Art von Algorithmus unterscheidet sich vom hierarchischen Clustern sehr, da ihm im Voraus mitgeteilt wird, wie viele verschiedene Cluster zu erzeugen sind. Der Algorithmus bestimmt dann die Größe der Cluster auf Basis der Datenstruktur.

Die K-Means-Clusteranalyse beginnt mit *k* zufällig platzierten *Clusterzentren* (*Centroids*, Positionen im Raum, die den Mittelpunkt des Clusters bestimmen) und weist jedes Element dem nächstliegenden Zentrum zu. Nach der Zuweisung werden die Clusterzentren zur gemittelten Position aller Knoten verschoben, die ihnen zugewiesen sind, und die Zuweisungen werden erneut durchgeführt. Dieser Prozess wiederholt sich, bis sich die Zuweisungen nicht mehr ändern. Abbildung 3-5 zeigt diesen Prozess für fünf Elemente und zwei Cluster.

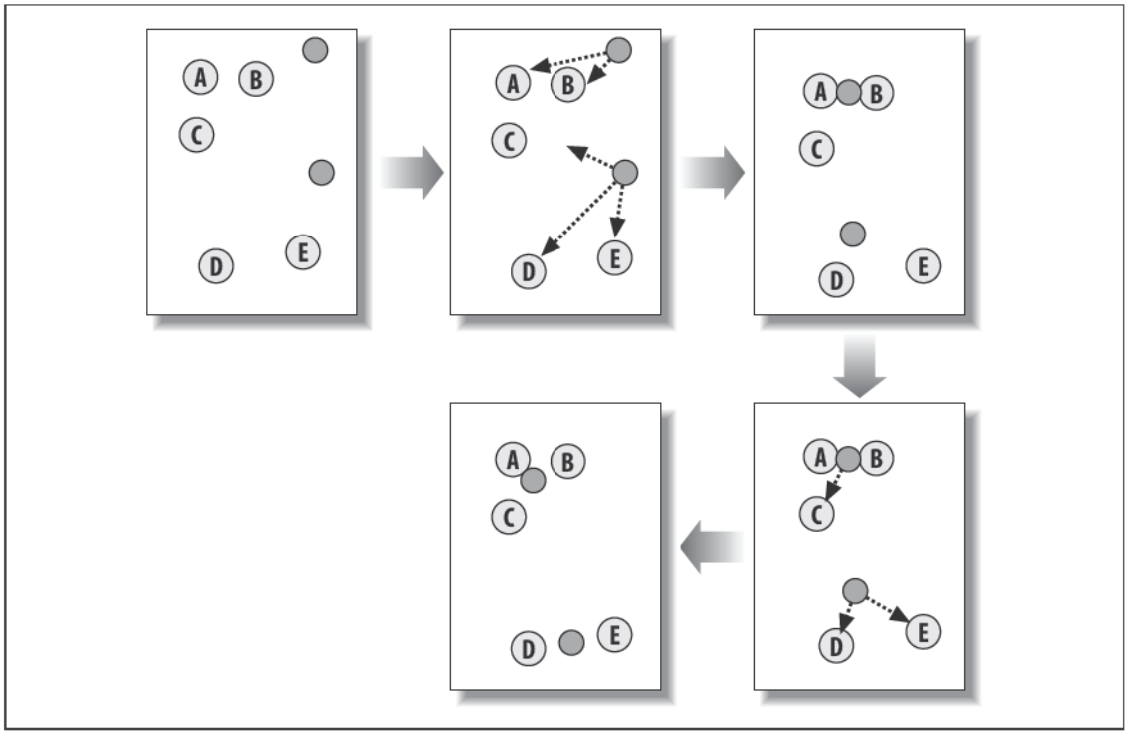

Abbildung 3-5: K-Means-Clusteranalyse mit zwei Clustern

Im ersten Kasten sind die beiden Clusterzentren (die schwarzen Kreise) zufällig positioniert. Kasten 2 zeigt, dass jedes Element dem nächsten Zentrum zugewiesen wird – in diesem Fall gehen A und B an das obere Clusterzentrum und C, D und E an das untere. Im dritten Kasten wurde jedes Clusterzentrum an die gemittelte Position seiner Elemente verschoben. Nachdem die Zuweisungen erneut durchgeführt

e-bol.net

wurden, zeigt sich, dass C nun näher am oberen Clusterzentrum liegt, während D und E beim unteren bleiben. Daher ist das abschließende Ergebnis nun, dass A, B und C zu einem Cluster gehören und D und E zum anderen.

Die Funktion für die K-Means-Clusteranalyse erwartet die gleichen Datenzeilen als Eingabe wie der hierarchische Clusteralgorithmus, dazu die Anzahl an Clustern (k), die der Aufrufende erhalten möchte. Fügen Sie *clusters.py* diesen Code hinzu:

```
import random

def kcluster(rows,distance=pearson,k=4):
  # Minimale und maximale Werte für jeden Punkt bestimmen.
  ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
  for i in range(len(rows[0]))]

  # k zufällig verteilte Clusterzentren erstellen.
  clusters=[[random.random( )*(ranges[i][1]-ranges[i][0])+ranges[i][0]
  for i in range(len(rows[0]))] for j in range(k)]

  lastmatches=None
  for t in range(100):
    print 'Iteration %d' % t
    bestmatches=[[] for i in range(k)]

    # Für jede Zeile das am nächsten gelegene Clusterzentrum finden.
    for j in range(len(rows)):
      row=rows[j]
      bestmatch=0
      for i in range(k):
        d=distance(clusters[i],row)
        if d<distance(clusters[bestmatch],row): bestmatch=i
      bestmatches[bestmatch].append(j)

    # Sind die Ergebnisse die gleichen wie beim letzten Durchlauf, abbrechen.
    if bestmatches==lastmatches: break
    lastmatches=bestmatches

    # Die Clusterzentren zur gemittelten Position ihrer Elemente verschieben.
    for i in range(k):
      avgs=[0.0]*len(rows[0])
      if len(bestmatches[i])>0:
        for rowid in bestmatches[i]:
          for m in range(len(rows[rowid])):
            avgs[m]+=rows[rowid][m]
        for j in range(len(avgs)):
          avgs[j]/=len(bestmatches[i])
        clusters[i]=avgs

  return bestmatches
```

Dieser Code erstellt zunächst zufällig verteilt eine Reihe von Clustern im Bereich jeder der Variablen. Mit jeder Iteration werden die Zeilen einem der Clusterzentren

e-bol.net

zugewiesen, und die Schwerpunktdaten werden danach neu auf die gemittelte Position aller zugehörigen Elemente gesetzt. Wenn die Zuweisungen genau so wie beim letzten Durchlauf sind, wird der Prozess beendet und die `k` Listen, jede für ein Cluster stehend, zurückgegeben. Die Anzahl an Iterationen, um zum Endergebnis zu kommen, ist im Vergleich zur hierarchischen Clusteranalyse ziemlich klein.

Da diese Funktion zu Beginn zufällig positionierte Clusterzentren verwendet, wird die Reihenfolge der zurückgegebenen Ergebnisse fast immer unterschiedlich sein. Es ist auch möglich, dass sich die Inhalte der Cluster pro Durchlauf unterscheiden, weil die initialen Positionen der Clusterzentren unterschiedlich waren.

Sie können diese Funktion mit der Blog-Datenmenge ausprobieren. Sie sollte deutlich schneller laufen als die hierarchische Clusteranalyse:

```
>> reload(clusters)
>> kclust=clusters.kcluster(data,k=10)
Iteration 0
...
>> [rownames[r] for r in k[0]]
['The Viral Garden', 'Copyblogger', 'Creating Passionate Users', 'Oilman',
'ProBlogger Blog Tips', "Seth's Blog"]
>> [rownames[r] for r in k[1]]
etc..
```

`kclust` enthält nun eine Liste mit IDs für jedes Cluster. Testen Sie die Clusteranalyse mit verschiedenen Werten für `k` aus, um zu sehen, was für einen Einfluss dies auf die Ergebnisse hat.

Cluster mit Vorlieben

Eine der schönen tSeiten des wachsenden Interesses an Social-Networking-Sites ist, dass große Datenmengen bereitstehen, die freiwillig von den Leuten beigetragen wurden. Eine solche Site heißt Zebo (*http://www.zebo.com*). Dort kann man ein Konto anlegen und dann Listen von Dingen erstellen, die man besitzt, sowie von Dingen, die man gern hätte. Aus Sicht eines Werbenden, aber auch eines Gesellschaftskritikers, sind das sehr interessante Informationen, da man mit ihnen den Weg bestimmen kann, über den sich Vorlieben ganz natürlich anordnen.

Beschaffen und Vorbereiten der Daten

Dieser Abschnitt wird sich mit dem Erstellen einer Datenmenge von der Zebo-Website befassen. Dazu gehört das Herunterladen und Parsen vieler Seiten der Site, um herauszubekommen, was jeder Benutzer gern haben möchte. Wenn Sie diesen Abschnitt lieber überspringen möchten, können Sie eine vorgefertigte Datenmenge unter *http://kiwitobes.com/clusters/zebo.txt* herunterladen.

e-bol.net

Beautiful Soup

Beautiful Soup ist eine ausgezeichnete Bibliothek zum Parsen von Webseiten und zum Aufbauen einer strukturierten Repräsentation. Sie ermöglicht Ihnen, beliebige Elemente auf der Seite über den Typ, die ID oder irgendeine seiner Eigenschaften zu finden und eine Textdarstellung seines Inhalts zu erstellen. Beatiful Soup geht zudem sehr tolerant mit Webseiten um, die kaputtes HTML enthalten, was besonders beim Erstellen von Daten aus Websites nützlich ist.

Sie können Beautiful Soup unter *http://crummy.com/software/BeautifulSoup* herunterladen. Die Bibliothek ist eine einzelne Python-Datei, die Sie in Ihren Python-Bibliothekspfad oder in den Pfad kopieren können, in dem Sie arbeiten und aus dem Sie den Python-Interpreter starten.

Sobald Sie Beautiful Soup installiert haben, können Sie sie mit Ihrem Interpreter ausprobieren:

```
>> import urllib2
>> from BeautifulSoup import BeautifulSoup
>> c=urllib2.urlopen('http://kiwitobes.com/wiki/Programming_language.html')
>> soup=BeautifulSoup(c.read())
>> links=soup('a')
>> links[10]
<a href="/wiki/Algorithm.html" title="Algorithm">algorithms</a>
>> links[10]['href']
u'/wiki/Algorithm.html'
```

Um eine *Soup* zu erzeugen, also die Repräsentation einer Webseite mit Beatiful Soup, initialisieren Sie sie einfach mit dem Inhalt der Seite. Sie können die Soup mit einem Tag-Typ, zum Beispiel `a`, initialisieren und erhalten dann eine Liste von Objekten dieses Typs. Jedes dieser Objekte ist auch adressierbar und ermöglicht Ihnen damit, direkt auf Eigenschaften und andere Objekte, die in der Hierarchie darunter stehen, zuzugreifen.

Web Scraping der Zebo-Ergebnisse

Die Struktur der Suchseite bei Zebo ist ziemlich komplex, aber man kann leicht herausfinden, welche Teile der Seite die Auflistung der Elemente darstellen, da alle die Klasse `bgverdanasmall` haben. Sie können dies zu Ihrem Vorteil nutzen, um die wichtigen Daten von der Seite zu holen. Erstellen Sie eine neue Datei mit dem Namen *downloadzebodata.py* und fügen Sie den folgenden Code ein:

```
from BeautifulSoup import BeautifulSoup
import urllib2
import re
chare=re.compile(r'[!-\.&]')
itemowners={}
```

e-bol.net

```
# zu entfernende Wörter
dropwords=['a','new','some','more','my','own','the','many','other','another']

currentuser=0
for i in range(1,51):
  # URL für die Suchseite mit Wünschen
  c=urllib2.urlopen(
  'http://member.zebo.com/Main?event_key=USERSEARCH&wiowiw=wiw&keyword=car&page=%d'
  % (i))
  soup=BeautifulSoup(c.read( ))
  for td in soup('td'):
    # Tabellenfelder mit der Klasse bgverdanasmall finden.
    if ('class' in dict(td.attrs) and td['class']=='bgverdanasmall'):
      items=[re.sub(chare,'',a.contents[0].lower()).strip( ) for a in td('a')]
      for item in items:
        # Zusätzliche Wörter entfernen.
        txt=' '.join([t for t in item.split(' ') if t not in dropwords])
        if len(txt)<2: continue
        itemowners.setdefault(txt,{})
        itemowners[txt][currentuser]=1
      currentuser+=1
```

Dieser Code lädt die ersten 50 Seiten der Suchergebnisse mit Wünschen von Zebo herunter und parst sie. Da alle Elemente als freier Text eingegeben werden, gibt es eine Menge auszumisten. Unter anderem müssen Wörter wie »a« und »some« entfernt, Satzzeichen gelöscht und alles in Kleinbuchstaben umgewandelt werden.

Ist das erledigt, muss der Code als Nächstes eine Liste der Dinge erstellen, die mehr als fünf Leute haben wollen, dann muss er eine Matrix mit anonymisierten Benutzern als Spalten und den Gegenständen als Zeilen aufbauen und schließlich diese Matrix in eine Datei schreiben. Fügen Sie folgende Zeilen am Ende von *downloadzebodata.py* ein:

```
out=file('zebo.txt','w')
out.write('Item')
for user in range(0,currentuser): out.write('\tU%d' % user)
out.write('\n')
for item,owners in itemowners.items( ):
  if len(owners)>10:
    out.write(item)
    for user in range(0,currentuser):
      if user in owners: out.write('\t1')
      else: out.write('\t0')
    out.write('\n')
```

Führen Sie den unten stehenden Befehl an der Befehlszeile aus, um eine Datei mit dem Namen *zebo.txt* zu erzeugen, die das gleiche Format wie die Blog-Daten hat. Der einzige Unterschied liegt darin, dass hier statt Wortfrequenzen eine 1 steht, wenn eine Person einen bestimmten haben Gegenstand möchte, und eine 0, wenn nicht:

```
c:\code\cluster>python downloadzebodata.py
```

e-bol.net

Definieren einer Abstandsmetrik

Der Pearson-Korrelationskoeffizient funktioniert bei den Blog-Daten gut, weil es sich bei den Werten um Wortfrequenzen handelt. Dieser Datenbestand hier enthält aber nur Einsen und Nullen für die Anwesenheit oder Abwesenheit eines Wunschs. Es wäre also sinnvoller, ein Überlappungsmaß zwischen Personen zu definieren, die zwei Dinge haben wollen. Dafür gibt es ein Maß namens *Tanimoto*-Koeffizient, das dem Verhältnis zwischen der Schnittmenge (da sind nur die Gegenstände, die sich in beiden Mengen befinden) zur Vereinigungsmenge (alle Elemente in beiden Mengen) entspricht. Dies lässt sich für zwei Vektoren leicht definieren:

```
def tanamoto(v1,v2):
  c1,c2,shr=0,0,0

  for i in range(len(v1)):
    if v1[i]!=0: c1+=1 # in v1
    if v2[i]!=0: c2+=1 # in v2
    if v1[i]!=0 and v2[i]!=0: shr+=1 # in both

  return 1.0-(float(shr)/(c1+c2-shr))
```

Damit wird ein Wert zwischen 1,0 und 0,0 zurückgegeben. Der Wert 1,0 gibt an, dass keiner, der sich das erste Element wünscht, auch das zweite will, während 0,0 bedeutet, dass genau die gleichen Leute sowohl das erste als auch das zweite Element haben möchten.

Ergebnisse der Clusteranalyse

Da die Daten im gleichen Format wie zuvor vorhanden sind, können auch die gleichen Funktionen genutzt werden, um die hierarchischen Cluster zu erzeugen und zu zeichnen; fügen Sie sie *clusters.py* hinzu:

```
>> reload(clusters)
>> wants,people,data=clusters.readfile('zebo.txt')
>> clust=clusters.hcluster(data,distance=clusters.tanamoto)
>> clusters.drawdendrogram(clust,wants)
```

Damit wird eine neue Datei *clusters.jpg* erzeugt, die die Cluster mit den gewünschten Dingen darstellt. Das Ergebnis für die herunterladbaren Daten ist in Abbildung 3-6 zu sehen. Es gibt keine weltbewegenden neuen Erkenntnisse – die gleichen Leute wollen eine Xbox, eine PlayStation Portable und eine PlayStation 3 –, aber es gibt ein paar eindeutige Gruppen, die herausstechen, so zum Beispiel die sehr anspruchsvollen Menschen (Schiff, Flugzeug, Insel) und die eher an inneren Werten interessierten (Freunde, Liebe, Glück). Es ist auch interessant festzustellen, dass Menschen, die »Geld« wollen, sich nur ein »Haus« wünschen, während diejenigen, die »viel Geld« wollen, eher ein »nettes Haus« möchten.

Indem man die ursprüngliche Suche ändert, die Anzahl der zu ladenden Seiten anpasst oder die »Besitz«-Daten statt der »Wunsch«-Daten nutzt, können Sie ver-

e-bol.net

mutlich andere interessante Gruppen von Elementen entdecken. Sie können ebenfalls versuchen, die Matrix zu transponieren, und die Benutzer gruppieren, was noch interessanter wäre, wenn man auch das Alter der Leute dazunimmt, um zu sehen, wie das Alter die Menschen unterscheidet.

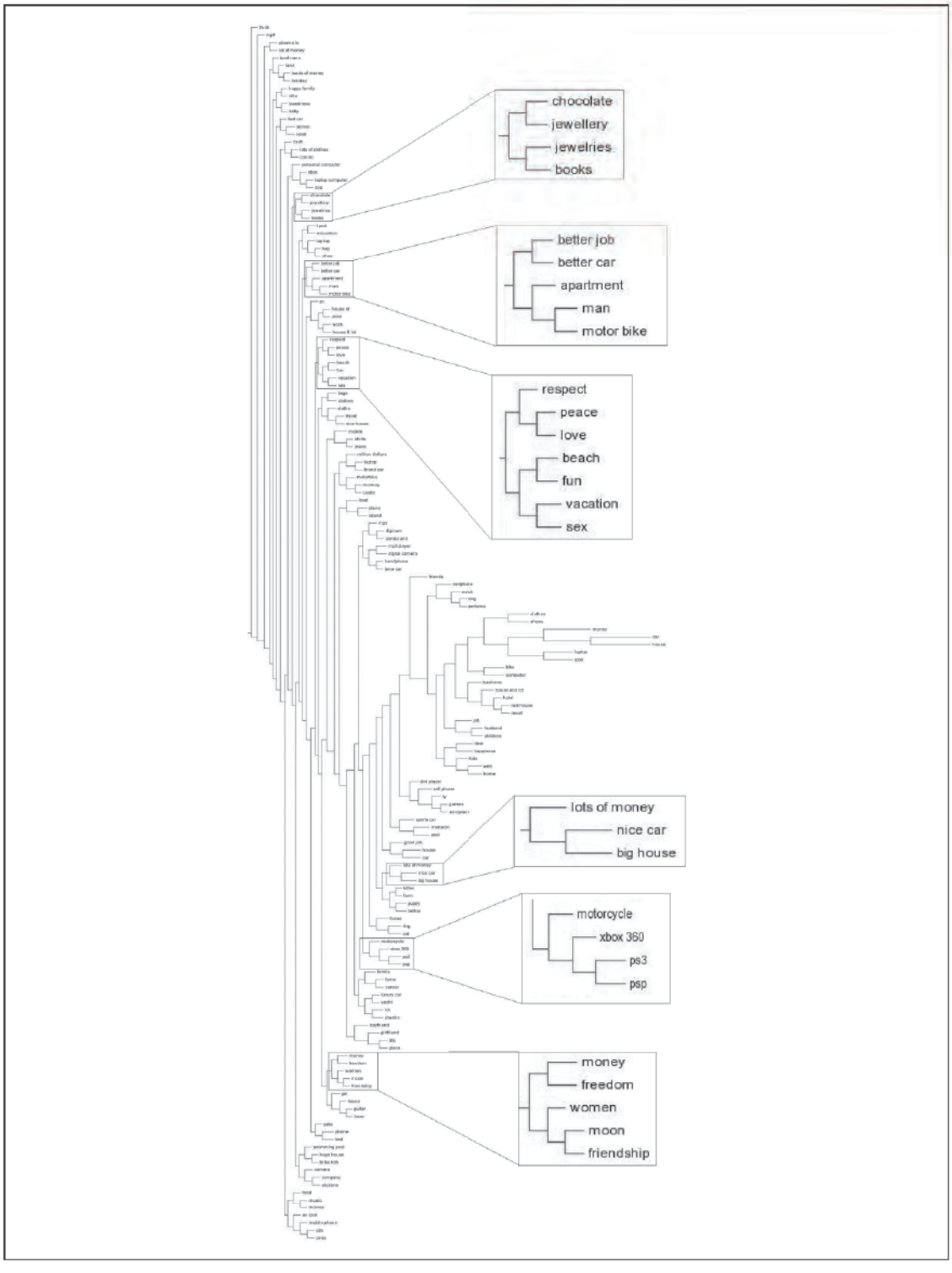

Abbildung 3-6: Cluster mit Dingen, die die Menschen haben wollen

e-bol.net

Daten in zwei Dimensionen betrachten

Die Clusteralgorithmen in diesem Kapitel wurden von Darstellungen der Daten in zwei Dimensionen begleitet, wobei der Abstand der verschiedenen Objekte dadurch repräsentiert wird, wie weit sie voneinander entfernt sind. Da die meisten Beispiele aus dem wahren Leben, die Sie in Cluster einteilen würden, mehr als zwei Parameter besitzen, könnten Sie die Daten nicht einfach nehmen und als Graph aufzeichnen. Aber um das Verhältnis zwischen den verschiedenen Objekten zu verstehen, wäre es sehr nützlich, sie auf einer Seite gegeneinander aufzuzeichnen, wobei ein geringerer Abstand für eine größere Ähnlichkeit stünde.

Dieser Abschnitt wird eine Technik namens *multidimensionale Skalierung* vorstellen, die genutzt wird, um eine zweidimensionale Darstellung der Daten zu finden. Der Algorithmus nimmt den Unterschied zwischen jedem Paar von Elementen und versucht, einen Graphen zu erstellen, in dem die Abstände zwischen den Elementen zu deren Unterschieden passen. Dazu rechnet der Algorithmus zunächst die Zielabstände zwischen allen Elementen aus. Im Blog-Datenbestand wurde der Pearson-Korrelationskoeffizient genutzt, um die Elemente zu vergleichen. Ein Beispiel dafür findet sich in Tabelle 3-2.

Tabelle 3-2: Beispiel-Abstands-Matrix

	A	B	C	D
A	0,0	0,2	0,8	0,7
B	0,2	0,0	0,9	0,8
C	0,8	0,9	0,0	0,1
D	0,7	0,8	0,1	0,0

Als Nächstes werden alle Elemente (in diesem Fall die Blogs) zufällig auf dem zweidimensionalen Graphen positioniert, wie in Abbildung 3-7 zu sehen.

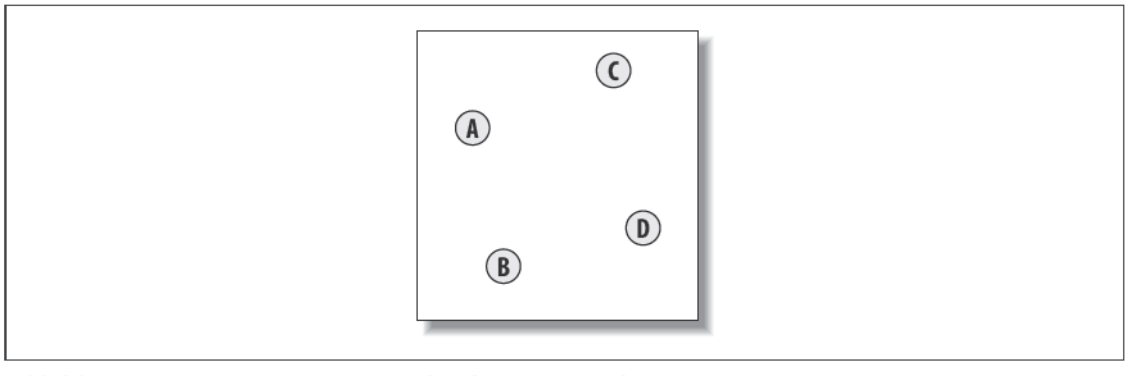

Abbildung 3-7: Ausgangspositionen für die 2-D-Projektion

e-bol.net

Die aktuellen Abstände zwischen allen Elementen werden mit dem tatsächlichen Abstand ermittelt (der Summe der Differenzen der Quadrate), wie in Abbildung 3-8 gezeigt.

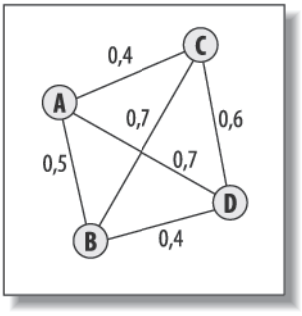

Abbildung 3-8: Abstände zwischen Elementen

Für jedes Elementpaar wird der Zielabstand mit dem aktuellen Abstand verglichen, und ein Fehlerwert wird ermittelt. Jedes Element wird dann ein bisschen näher heran- oder etwas weiter wegbewegt – in Abhängigkeit vom Fehlerwert zwischen den beiden Elementen. Abbildung 3-9 zeigt die Kräfte, die auf Element A wirken. Der Abstand zwischen A und B ist im Graphen 0,5, die Zieldistanz ist aber nur 0,2, daher muss A näher an B bewegt werden. Gleichzeitig wird A auch von C und D wegbewegt, da es sich zu nahe daran befindet.

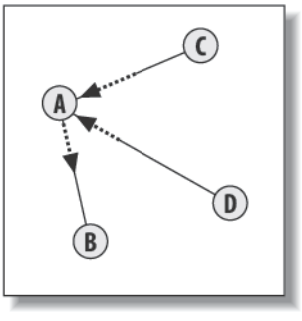

Abbildung 3-9: Kräfte, die auf A wirken

Jeder Knoten wird entsprechend der Kombination aller anderen Knoten, die an ihm ziehen oder ihn schieben, bewegt. Jedes Mal, wenn das passiert, wird der Unterschied zwischen dem aktuellen Abstand und dem Zielabstand ein wenig geringer. Diese Prozedur wird so oft wiederholt, bis die Gesamtsumme aller Fehler nicht mehr durch ein Verschieben der Knoten reduziert werden kann.

Die Funktion dafür übernimmt den Datenvektor und liefert einen mit nur zwei Spalten zurück – den x- und y-Koordinaten der Elemente im zweidimensionalen Graphen. Fügen Sie diese Funktion der Datei *clusters.py* hinzu:

e-bol.net

```
def scaledown(data,distance=pearson,rate=0.01):
  n=len(data)

  # der wirkliche Abstand zwischen jedem Elementpaar
  realdist=[[distance(data[i],data[j]) for j in range(n)]
            for i in range(0,n)]

  outersum=0.0

  # zufällig initialisierte Ausgangspunkte für die Positionen in 2-D
  loc=[[random.random(),random.random( )] for i in range(n)]
  fakedist=[[0.0 for j in range(n)] for i in range(n)]

  lasterror=None
  for m in range(0,1000):
    # Momentane Abstände finden.
    for i in range(n):
      for j in range(n):
        fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
                                 for x in range(len(loc[i]))]))

    # Punkte bewegen.
    grad=[[0.0,0.0] for i in range(n)]

    totalerror=0
    for k in range(n):
      for j in range(n):
        if j==k: continue
        # Der Fehler ist der prozentuale Unterschied zwischen den Abständen.
        errorterm=(fakedist[j][k]-realdist[j][k])/realdist[j][k]

        # Jeder Punkt muss zum anderen Punkt hin- oder von ihm wegbewegt
        # werden - abhängig vom Fehler.
        grad[k][0]+=((loc[k][0]-loc[j][0])/fakedist[j][k])*errorterm
        grad[k][1]+=((loc[k][1]-loc[j][1])/fakedist[j][k])*errorterm

        # Gesamtfehler merken
        totalerror+=abs(errorterm)
    print totalerror

    # Wenn das Ergebnis schlechter wird, sind wir fertig.
    if lasterror and lasterror<totalerror: break
    lasterror=totalerror

    # Jeden Punkt um die Lernrate mal den Gradienten verschieben.
    for k in range(n):
      loc[k][0]-=rate*grad[k][0]
      loc[k][1]-=rate*grad[k][1]

  return loc
```

e-bol.net

Um sich das anzeigen zu lassen, können Sie die PIL erneut nutzen und ein Bild mit allen Bezeichnern aller Elemente erstellen, die an den neuen Koordinaten dieser Elemente ausgegeben werden.

```
def draw2d(data,labels,jpeg='mds2d.jpg'):
  img=Image.new('RGB',(2000,2000),(255,255,255))
  draw=ImageDraw.Draw(img)
  for i in range(len(data)):
    x=(data[i][0]+0.5)*1000
    y=(data[i][1]+0.5)*1000
    draw.text((x,y),labels[i],(0,0,0))
  img.save(jpeg,'JPEG')
```

Zum Ausführen dieses Algorithmus rufen Sie `scaledown` auf, um die zweidimensionale Datenmenge zu erhalten, und dann `draw2d`, um sie zu zeichnen:

```
>> reload(clusters)
>> blognames,words,data=clusters.readfile('blogdata.txt')
>> coords=clusters.scaledown(data)
...
>> clusters.draw2d(coords,blognames,jpeg='blogs2d.jpg')
```

Abbildung 3-10 zeigt das Ergebnis des multidimensionalen Skalieralgorithmus. Die Cluster springen einem nicht so offensichtlich ins Auge wie beim Dendrogramm, aber es gibt trotzdem ein paar deutliche Themengruppen, wie zum Beispiel die Such-Engine-bezogenen am oberen Rand. Sie sind ziemlich weit weg von den politischen oder Promi-Blogs gelandet. Könnte man diese Repräsentation in drei Dimensionen anzeigen, würden die Cluster noch besser hervortreten, aber das wäre natürlich auf Papier etwas problematisch darzustellen.

Andere Dinge in der Clusteranalyse

Dieses Kapitel hat sich mit zwei Datenbeständen befasst, aber es gibt vieles andere, womit man arbeiten kann. Die *del.icio.us*-Daten aus Kapitel 2 können auch geclustert werden, um Benutzer- oder Lesezeichen-Gruppen zu finden. Auf die gleiche Art und Weise, wie die Blog-Feeds in Wortvektoren umgewandelt wurden, lassen sich alle heruntergeladenen Seiten auf ihre Wörter reduzieren.

Diese Ideen können auf viele verschiedene Bereiche angewandt werden, um interessante Dinge herauszufinden – Foren basierend auf der Wortfrequenz, Firmen aus Yahoo! Finance basierend auf verschiedensten Statistiken oder die Topkritiker bei Amazon aufgrund ihrer Vorlieben. Es wäre auch interessant, sich große Social Networks wie MySpace anzuschauen und die Personen aufgrund ihrer Freunde zu gruppieren oder eventuell andere Informationen zu nutzen, die sie über sich selbst bereitstellen (Lieblingsbands, Essen und so weiter).

Das Konzept, Elemente im Raum aufgrund ihrer Parameter darzustellen, wird in diesem Buch immer wieder auftauchen. Die multidimensionale Skalierung ist eine effektive Möglichkeit, eine Datenmenge sich so anzeigen zu lassen, dass sie leicht

e-bol.net

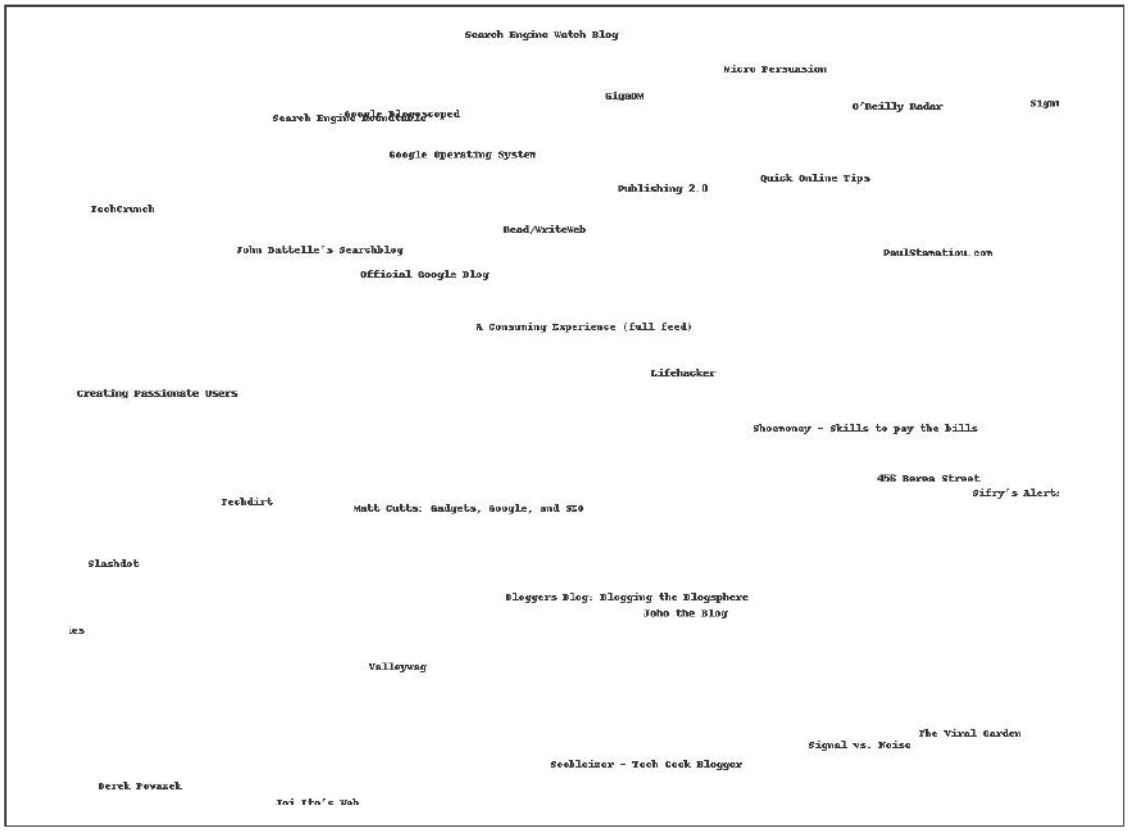

Abbildung 3-10: Teil der 2-D-Repräsentation des Blog-Raums

interpretiert werden kann. Es ist wichtig, sich darüber im Klaren zu sein, dass beim Skalieren einiges an Informationen verloren geht, aber das Ergebnis sollte Ihnen dabei helfen, die Algorithmen besser zu verstehen.

Übungen

1. Erstellen Sie mithilfe der *del.icio.us*-API aus Kapitel 2 eine Datenmenge mit Lesezeichen, auf die man eine Clusteranalyse anwenden kann. Lassen Sie die hierarchische und die K-Means-Clusteranalyse dafür laufen.
2. Passen Sie den Code zum Parsen der Blogs an, um einzelne Einträge statt ganzer Blogs zu clustern. Passen die Einträge eines Blogs in ein Cluster? Wie sieht es mit Einträgen eines Tages aus?
3. Versuchen Sie, den echten (Pythagoras-)Abstand beim Clustern der Blogs zu nutzen. Wie wirkt sich das auf die Ergebnisse aus?
4. Finden Sie heraus, was der Manhattan-Abstand ist. Erstellen Sie eine Funktion dafür und prüfen Sie, wie sich die Ergebnisse für die Zebo-Datenmenge damit ändern.

e-bol.net

5. Passen Sie die Funktion für die K-Means-Clusteranalyse an, damit sie neben den Clusterergebnissen auch den Gesamtabstand zwischen allen Elementen und ihren zugehörigen Clusterzentren zurückgibt.
6. Erstellen Sie nach Abschluss von Übung 5 eine Funktion, die die K-Means-Clusteranalyse für verschiedene Werte von k ausführt. Wie ändert sich der Gesamtabstand, wenn die Anzahl der Cluster zunimmt? An welchem Punkt wird die Verbesserung durch mehr Cluster sehr gering?
7. Multidimensionale Skalierung in zwei Dimensionen ist sehr einfach auszugeben, aber man kann sie auch mit beliebig vielen Dimensionen durchführen. Versuchen Sie, den Code so anzupassen, dass er mit einer Dimension (alle Punkte auf einer Linie) funktioniert. Nun versuchen Sie, ihn für drei Dimensionen zum Laufen zu bringen.

e-bol.net

KAPITEL 4

Suchen und Bewerten

Dieses Kapitel dreht sich um Volltext-Such-Engines, die es ermöglichen, große Mengen an Dokumenten nach einer Liste von Wörtern zu durchsuchen und die Ergebnisse nach ihrer Relevanz bezüglich dieser Wörter zu ordnen. Algorithmen für Volltextsuchen gehören zu den wichtigsten im Bereich der kollektiven Intelligenz, und viele haben schon ihr Auskommen gefunden, indem sie hier neue Ideen entwickelten. Es wird allgemein davon ausgegangen, dass das rapide Wachstum von Google von einem akademischen Projekt zur weltweit beliebtesten Suchmaschine zum großen Teil auf dem PageRank-Algorithmus basiert – eine abgewandelte Version davon werden Sie in diesem Kapitel kennenlernen.

Die Suche nach Informationen ist ein umfangreiches Gebiet mit einer langen Geschichte. Dieses Kapitel wird nur ein paar der wichtigsten Konzepte vorstellen können, aber wir werden den Aufbau einer Such-Engine besprechen, die eine Menge von Dokumenten indexieren kann, und Ihnen damit die Möglichkeit geben, sich Gedanken über weitere Verbesserungen zu machen. Auch wenn der Fokus auf Such- und Bewertungsalgorithmen liegt und weniger auf den Infrastrukturanforderungen, die entstehen, wenn man große Teile des Web indexieren will, sollten Sie keine Probleme mit Datenmengen von bis zu 100.000 Seiten haben. In diesem Kapitel werden Sie alle notwendigen Schritte kennenlernen, um eine Reihe von Seiten zu analysieren, zu indexieren und zu durchsuchen und dazu die Ergebnisse auf viele verschiedene Arten einzustufen.

Was gehört zu einer Such-Engine?

Der erste Schritt beim Aufbauen einer Such-Engine ist, eine Möglichkeit zu entwickeln, die Dokumente zusammenzutragen. In manchen Fällen gehört dazu das *Crawlen* (was bedeutet, mit einer kleinen Menge an Dokumenten anzufangen und den Links von dort zu anderen Dokumenten zu folgen), in anderen Fällen beginnt man mit einer festen Sammlung von Dokumenten, eventuell aus einem Firmen-Intranet.

e-bol.net

Haben Sie diese Dokumente zusammengetragen, müssen sie indexiert werden. Dazu gehört normalerweise das Erstellen einer großen Tabelle mit den Dokumenten und Positionen aller Wörter. Je nach Anwendung müssen die Dokumente selbst nicht zwingend in einer Datenbank abgelegt sein – der Index muss dann nur eine Referenz (wie einen Pfad im Dateisystem oder eine URL) auf ihre Ablageorte enthalten.

Der letzte Schritt ist natürlich, eine bewertete Liste der Dokumente für eine Abfrage zurückzugeben. Das Finden jedes Dokuments mit einem bestimmten Satz an Wörtern ist recht einfach, wenn Sie erst einmal einen Index haben, aber die wahre Kunst liegt darin, wie die Ergebnisse sortiert sind. Es können viele Metriken erzeugt werden, und es gibt viele verschiedene Wege, sie so anzupassen, dass sich die Sortierreihenfolge ändert. Wenn Sie alle Metriken kennengelernt haben, werden Sie sich nur wünschen, dass die großen Such-Engines Ihnen mehr Kontrolle ermöglichen (»Warum kann ich Google nicht mitteilen, dass meine Wörter nahe beieinanderstehen müssen?«). Dieses Kapitel wird sich verschiedene Metriken anschauen, die auf dem Inhalt der Seite basieren, wie zum Beispiel die Wortfrequenz, sowie Metriken, die auf Informationen aufbauen, die nicht zum Inhalt der Seiten gehören, wie zum Beispiel den PageRank-Algorithmus, der berücksichtigt, wie andere Seiten auf die fragliche Seite verweisen.

Schließlich werden Sie ein *neuronales Netz* zum Bewerten der Abfragen erstellen. Das neuronale Netz wird lernen, Suchen mit Ergebnissen zu verbinden – basierend auf den Links, die die Menschen anklicken, nachdem sie eine Liste mit Suchergebnissen erhalten haben. Das neuronale Netz wird diese Informationen nutzen, um die Reihenfolge der Ergebnisse anzupassen, und damit darauf reagieren, was die Leute in der Vergangenheit angeklickt haben.

Um die Beispiele in diesem Kapitel durchzuarbeiten, werden Sie ein Python-Modul namens *searchengine* anlegen, das zwei Klassen hat: eine zum Crawlen und zum Erstellen der Datenbank und die andere für Volltextsuchen durch Abfragen der Datenbank. Die Beispiele werden *SQLite* nutzen, aber sie können leicht für eine klassische Client-Server-Datenbank angepasst werden.

Zu Beginn erstellen Sie eine neue Datei namens *searchengine.py* und fügen die folgende Klasse `crawler` und die Methodensignaturen ein, die Sie im Laufe dieses Kapitels mit Leben füllen werden:

```
class crawler:
  # Den Crawler mit dem Namen der Datenbank initialisieren.
  def __init__(self,dbname):
    pass

  def __del__(self):
    pass
  def dbcommit(self):
    pass
```

e-bol.net

```
# Hilfsfunktion, um eine Entry-ID zu erhalten und sie hinzuzufügen,
# wenn es sie noch nicht gibt.
def getentryid(self,table,field,value,createnew=True):
  return None

# Eine einzelne Seite indexieren.
def addtoindex(self,url,soup):
  print 'Indexiere %s' % url

# Den Text aus einer HTML-Seite holen (ohne Tags).
def gettextonly(self,soup):
  return None

# Die Wörter an den Trennzeichen aufteilen.
def separatewords(self,text):
  return None

# True zurückgeben, wenn diese URL schon indexiert ist.
def isindexed(self,url):
  return False

# Einen Link zwischen zwei Seiten hinzufügen.
def addlinkref(self,urlFrom,urlTo,linkText):
  pass

# Mit einer Liste von Seiten beginnen, bis zur gegebenen Tiefe
# in die Breite gehen und dabei Seiten indexieren.
def crawl(self,pages,depth=2):
  pass

# Die Datenbanktabellen anlegen.
def createindextables(self):
  pass
```

Ein einfacher Crawler

Ich gehe jetzt einmal davon aus, dass Sie keine große Sammlung an HTML-Dokumenten haben, die auf Ihrer Festplatte herumlungern und darauf warten, dass sie indexiert werden, daher werde ich Ihnen zeigen, wie Sie einen einfachen Crawler erstellen können. Er wird mit einer kleinen Zahl an Seiten gefüttert werden, die zu indexieren sind, und dann allen Links auf diesen Seiten folgen, um andere Seiten zu finden, deren Links er auch folgen wird. Dieser Prozess wird als Crawling oder *Spidering* bezeichnet.

Dazu muss Ihr Code die Seiten herunterladen, sie an den Indexer übergeben (den Sie im nächsten Abschnitt bauen) und dann parsen, um alle Links zu finden, die Sie als Nächstes ansteuern werden. Glücklicherweise gibt es eine Reihe von Bibliotheken, die Ihnen dabei helfen können.

e-bol.net

Für die Beispiele in diesem Kapitel habe ich Ihnen eine Kopie mehrerer Tausend Dateien von Wikipedia zusammengestellt, die auf *http://kiwitobes.com/wiki* zu finden sind.

Sie können den Crawler auf eine beliebige Menge an Seiten loslassen, aber Sie können auch die von meiner Site nutzen, wenn Sie Ihre Ergebnisse mit denen in diesem Kapitel vergleichen wollen.

urllib2 verwenden

urllib2 ist eine Bibliothek, die mit Python mitkommt und es Ihnen erleichtert, Seiten herunterzuladen – Sie müssen dazu nur die URL angeben. Sie werden diese Bibliothek in diesem Abschnitt nutzen, um die Seiten herunterzuladen, die indexiert werden. Um das einmal auszuprobieren, starten Sie Ihren Python-Interpreter und versuchen das Folgende:

```
>> import urllib2
>> c=urllib2.urlopen('http://kiwitobes.com/wiki/Programming_language.html')
>> contents=c.read()
>> print contents[0:50]
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Trans'
```

Alles, was Sie jetzt noch tun müssen, um den HTML-Code einer Seite in einem String zu speichern, ist der Aufbau einer Verbindung und das Lesen des Inhalts.

Crawler-Code

Der Crawler wird die Beautiful Soup-API nutzen, die in Kapitel 3 vorgestellt wurde – eine ausgezeichnete Bibliothek, die eine strukturierte Repräsentation von Webseiten aufbaut. Sie ist sehr tolerant gegenüber Webseiten mit kaputtem HTML, was sehr nützlich sein kann, wenn Sie einen Crawler aufbauen, da Sie nie wissen können, was für Seiten Ihnen unterkommen. Mehr Informationen zum Herunterladen und Installieren von Beautiful Soup finden Sie in Anhang A.

Mit urllib2 und Beautiful Soup können Sie einen Crawler aufbauen, der eine Liste mit URLs entgegennimmt, die indexiert werden sollen, und deren Links abklappert, um andere Seiten zu finden, die auch indexiert werden sollen. Als Erstes fügen Sie diese `import`-Anweisungen am Anfang von *searchengine.py* ein:

```
import urllib2
from BeautifulSoup import *
from urlparse import urljoin

# Eine Liste mit Wörtern erstellen, die zu ignorieren sind.
ignorewords=set(['the','of','to','and','a','in','is','it'])
```

Jetzt können Sie den Code für die Crawler-Funktion einfügen. Tatsächlich speichert er nicht das, was er gerade abgrast, aber er gibt bei seinen Wanderungen die URLs

e-bol.net

aus, sodass Sie prüfen können, ob er funktioniert. Sie müssen dies an das Ende der Datei anfügen (sodass es Teil der Klasse crawler ist):

```
def crawl(self,pages,depth=2):
  for i in range(depth):
    newpages=set( )
    for page in pages:
      try:
        c=urllib2.urlopen(page)
      except:
        print "Konnte %s nicht öffnen" % page
        continue
      soup=BeautifulSoup(c.read( ))
      self.addtoindex(page,soup)

      links=soup('a')
      for link in links:
        if ('href' in dict(link.attrs)):
          url=urljoin(page,link['href'])
          if url.find("'")!=-1: continue
          url=url.split('#')[0]  # location entfernen
          if url[0:4]=='http' and not self.isindexed(url):
            newpages.add(url)
          linkText=self.gettextonly(link)
          self.addlinkref(page,url,linkText)

      self.dbcommit( )

    pages=newpages
```

Diese Funktion läuft über die Liste der Seiten und ruft für jede Seite addtoindex auf (momentan wird nur die URL ausgegeben, aber Sie werden der Funktion im nächsten Abschnitt noch mehr Leben einhauchen). Dann nutzt sie Beautiful Soup, um alle Links auf dieser Seite zu erhalten und deren URLs einem Set namens newpages hinzuzufügen. Am Ende der Schleife wird newpages zu pages, und der Prozess beginnt von Neuem.

Diese Funktion kann rekursiv definiert werden, sodass jeder Link die Funktion erneut aufruft, aber eine solche zu Beginn breit angelegte Suche ermöglicht später einfachere Anpassungen des Codes, sei es zum kontinuierlichen Crawlen oder zum Speichern einer Liste mit noch nicht indexierten Seiten für ein späteres Abarbeiten. So wird auch das Risiko vermieden, dass der Stack überläuft.

Sie können diese Funktion im Python-Interpreter ausprobieren (müssen sie aber natürlich nicht bis zu Ende laufen lassen – können Sie Strg-C, wenn es Sie langweilt):

```
>> import searchengine
>> pagelist=['http://kiwitobes.com/wiki/Perl.html']
>> crawler=searchengine.crawler('')
>> crawler.crawl(pagelist)
Indexing http://kiwitobes.com/wiki/Perl.html
Could not open http://kiwitobes.com/wiki/Module_%28programming%29.html
```

e-bol.net

```
Indexing http://kiwitobes.com/wiki/Open_Directory_Project.html
Indexing http://kiwitobes.com/wiki/Common_Gateway_Interface.html
```

Sie haben vielleicht bemerkt, dass manche Seiten wiederholt angesprungen werden. Es gibt im Code einen Platzhalter für eine andere Funktion, `isindexed`, die feststellt, ob eine Seite schon (kürzlich) indexiert wurde, bevor sie zu `newpages` hinzugefügt wird. Damit können Sie diese Funktion jederzeit auf jede Liste von URLs loslassen, ohne sich sorgen zu müssen, dass unnötige Arbeit getan wird.

Aufbauen des Index

Der nächste Schritt ist, die Datenbank für den Volltextindex einzurichten. Wie ich schon weiter oben erwähnt habe, ist der Index eine Liste mit allen verschiedenen Wörtern, zusammen mit den Dokumenten, in denen sie vorkommen, und ihrer Position darin. In diesem Beispiel werden Sie sich den eigentlichen Text auf der Seite anschauen und die Elemente ignorieren, die kein Text sind (zum Beispiel Tags). Zudem werden Sie die einzelnen Wörter indexieren, wobei alle Satzzeichen entfernt wurden. Die Methode zum Aufteilen der Wörter ist nicht perfekt, wird aber zum Aufbau einer einfachen Such-Engine ausreichend sein.

Da eine Behandlung der verschiedenen Datenbanksysteme oder das Einrichten eines Datenbankservers nicht Thema dieses Buchs ist, wird dieses Kapitel Ihnen zeigen, wie Sie den Index mit SQLite speichern können. SQLite ist eine embedded Datenbank, die sich sehr leicht einrichten lässt und eine ganze Datenbank in einer Datei ablegt. SQLite nutzt SQL für die Abfragen, daher sollte es nicht allzu schwierig sein, den Beispielcode für eine andere Datenbank anzupassen. Die Python-Implementierung hat den Namen pysqlite, und Sie können sie von *http://initd.org/tracker/pysqlite* herunterladen.

Es gibt dort einen Windows-Installer sowie Anweisungen, wie man sie auf anderen Betriebssystemen installiert. Anhang A enthält weitere Informationen dazu, wie man pysqlite herunterlädt und installiert.

Nachdem Sie SQLite installiert haben, fügen Sie diese Zeile am Anfang von *searchengine.py* hinzu:

```
from pysqlite2 import dbapi2 as sqlite
```

Sie müssen auch die Methoden `__init__`, `__del__` und `dbcommit` ändern, um die Datenbank zu öffnen und zu schließen:

```
def __init__(self,dbname):
  self.con=sqlite.connect(dbname)

def __del__(self):
  self.con.close( )

def dbcommit(self):
  self.con.commit( )
```

e-bol.net

Erstellen des Schemas

Führen Sie den Code jetzt noch nicht aus – Sie müssen erst die Datenbank vorbereiten. Das Schema für den grundlegenden Index besteht aus fünf Tabellen. Die erste Tabelle (`urllist`) enthält die Liste der URLs, die schon indexiert wurden. Die zweite Tabelle (`wordlist`) ist die Liste der Wörter und die dritte Tabelle (`wordlocation`) eine Liste der Positionen der Wörter in den Dokumenten. Die restlichen zwei Tabellen definieren die Links zwischen Dokumenten. Die Tabelle `link` speichert zwei URL-IDs, mit denen ein Link von einem Dokument zu einem anderen definiert wird, während `linkwords` die Spalten `wordid` und `linkid` verwendet, um abzuspeichern, welche Wörter tatsächlich in diesem Link genutzt werden. Das Schema ist in Abbildung 4-1 zu sehen.

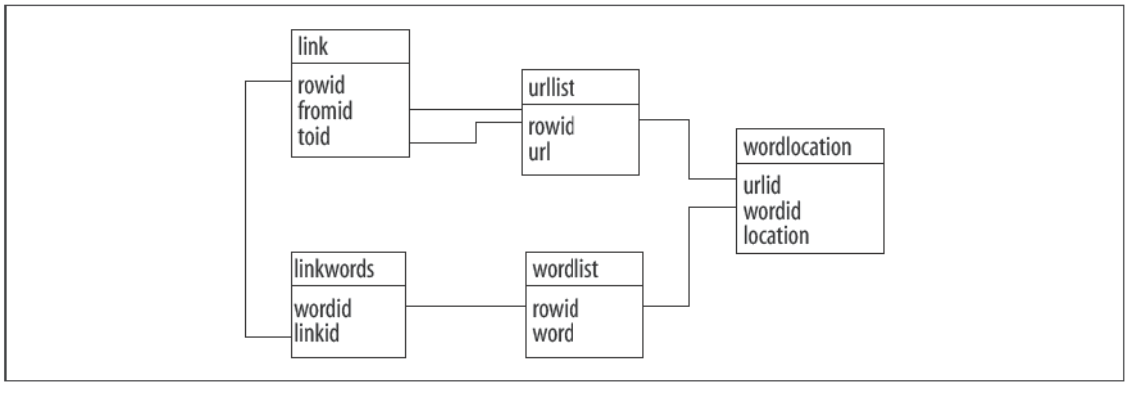

Abbildung 4-1: Schema für die Such-Engine

Alle Tabellen haben in SQLite standardmäßig ein Feld namens `rowid`, sodass Sie nicht explizit eine ID für diese Tabellen anlegen müssen. Um eine Funktion zum Erstellen aller dieser Tabellen zu erzeugen, fügen Sie diesen Code am Ende von *searchengine.py* ein, sodass er Teil der Klasse `crawler` ist:

```
def createindextables(self):
  self.con.execute('create table urllist(url)')
  self.con.execute('create table wordlist(word)')
  self.con.execute('create table wordlocation(urlid,wordid,location)')
  self.con.execute('create table link(fromid integer,toid integer)')
  self.con.execute('create table linkwords(wordid,linkid)')
  self.con.execute('create index wordidx on wordlist(word)')
  self.con.execute('create index urlidx on urllist(url)')
  self.con.execute('create index wordurlidx on wordlocation(wordid)')
  self.con.execute('create index urltoidx on link(toid)')
  self.con.execute('create index urlfromidx on link(fromid)')
  self.dbcommit()
```

Diese Funktion erstellt das Schema für alle von Ihnen verwendeten Tabellen und dazu ein paar Indizes zum Beschleunigen der Suche. Diese Indizes sind wichtig, da die Datenmengen potenziell sehr groß werden können. Geben Sie diese Befehle in Ihrer Python-Session ein, um eine Datenbank mit dem Namen *searchindex.db* zu erstellen:

e-bol.net

```
>> reload(searchengine)
>> crawler=searchengine.crawler('searchindex.db')
>> crawler.createindextables()
```

Später werden Sie das Schema um eine zusätzliche Tabelle ergänzen, um eine Bewertungsmetrik basierend auf der Anzahl eingehender Links ermöglichen zu können.

Die Wörter auf einer Seite finden

Die Dateien, die Sie aus dem Web laden, enthalten HTML-Code und damit eine Reihe von Tags, Eigenschaften und anderen Informationen, die nicht in den Index gehören. Als Erstes müssen Sie also alle Teile der Seite extrahieren, die Text sind. Sie können das tun, indem Sie die Beautiful Soup nach Textknoten durchsuchen und deren Inhalte sammeln. Fügen Sie diesen Code Ihrer Funktion `gettextonly` hinzu:

```
def gettextonly(self,soup):
  v=soup.string
  if v==None:
    c=soup.contents
    resulttext=''
    for t in c:
      subtext=self.gettextonly(t)
      resulttext+=subtext+'\n'
    return resulttext
  else:
    return v.strip()
```

Diese Funktion liefert einen langen String mit dem gesamten Text auf der Seite zurück, indem sie das HTML-Document-Object-Modell rekursiv durchläuft und nach Textknoten sucht. Text aus verschiedenen Abschnitten wird in unterschiedliche Absätze kopiert. Es ist wichtig, die Reihenfolge der Abschnitte für einige der Metriken zu bewahren, die Sie später berechnen werden.

Als Nächstes kommt die Funktion `separatewords` dran, die einen String in eine Liste einzelner Wörter aufteilt, sodass sie dem Index hinzugefügt werden können. Es ist nicht so einfach, es gründlich zu machen, wie Sie vielleicht denken mögen, und es wurde viel geforscht und ausprobiert, um die Technik zu verbessern. Aber für diese Beispiele wird es ausreichen, alles, was kein Buchstabe und keine Ziffer ist, als Trennzeichen zu behandeln. Sie können dies mit einem regulären Ausdruck erreichen. Ersetzen Sie die Definition von `separatewords` durch das Folgende:

```
def separatewords(self,text):
  splitter=re.compile('\\W*')
  return [s.lower() for s in splitter.split(text) if s!='']
```

Da diese Funktion alles, was nicht Buchstabe oder Ziffer ist, als Trenner ansieht, gibt es keine Probleme beim Extrahieren (englischer) Wörter, aber bei Begriffen wie

e-bol.net

»C++« arbeitet sie nicht zufriedenstellend (allerdings gibt es keinen Ärger bei »python«). Sie können mit dem regulären Ausdruck experimentieren, um die Funktionsweise bei verschiedene Arten von Suchen zu verbessern.

Eine andere Möglichkeit ist, bei den Wörtern die Suffixe mit einem *Stemming*-Algorithmus zu entfernen. Diese Algorithmen versuchen, die Wörter in ihre Wortstämme zurückzuführen. So wird das Wort »Indexieren« zum Beispiel zu »Index«, sodass Personen, die nach dem Wort »Index« suchen, auch Dokumente aufgeführt bekommen, die das Wort »Indexieren« enthalten. Dazu werden die Wörter sowohl beim Crawlen als auch beim Suchen auf ihren Wortstamm reduziert. Eine vollständige Behandlung des Stemmings passt nicht in den Rahmen dieses Kapitels, aber Sie können eine Python-Implementierung des bekannten *Porter-Stemmer-Algorithmus* unter *http://www.tartarus.org/~martin/PorterStemmer/index.html* bekommen.

Dem Index hinzufügen

Nun ist es an der Zeit, die Methode `addtoindex` mit Leben zu füllen. Diese Methode wird die zwei Funktionen aufrufen, die im vorigen Abschnitt definiert wurden, um eine Liste mit Wörtern auf der Seite zu erstellen. Dann wird sie die Seite und alle Wörter dem Index hinzufügen und Verknüpfungen zwischen ihnen mit den Positionen im Dokument anlegen. In diesem Beispiel handelt es sich bei der Position um den Index innerhalb der Wortliste.

Hier der Code für `addtoindex`:

```
def addtoindex(self,url,soup):
  if self.isindexed(url): return
  print 'Indexiere '+url

  # Einzelne Wörter erhalten.
  text=self.gettextonly(soup)
  words=self.separatewords(text)

  # URL-ID holen.
  urlid=self.getentryid('urllist','url',url)

  # Jedes Wort mit dieser URL verknüpfen.
  for i in range(len(words)):
    word=words[i]
    if word in ignorewords: continue
    wordid=self.getentryid('wordlist','word',word)
    self.con.execute("insert into wordlocation(urlid,wordid,location) \
      values (%d,%d,%d)" % (urlid,wordid,i))
```

Sie werden nun auch die Hilfsfunktion `getentryid` brauchen. Sie liefert einfach die ID eines Eintrags zurück. Wenn der Eintrag noch nicht existiert, wird er erstellt und die ID zurückgegeben:

e-bol.net

```
def getentryid(self,table,field,value,createnew=True):
  cur=self.con.execute(
  "select rowid from %s where %s='%s'" % (table,field,value))
  res=cur.fetchone()
  if res==None:
    cur=self.con.execute(
    "insert into %s (%s) values ('%s')" % (table,field,value))
    return cur.lastrowid
  else:
    return res[0]
```

Schließlich müssen Sie noch den Code für `isindexed` einfügen, der ermittelt, ob sich die Seite schon in der Datenbank befindet und, wenn ja, ob schon Wörter mit ihr verknüpft sind:

```
def isindexed(self,url):
  u=self.con.execute \
    ("select rowid from urllist where url='%s'" % url).fetchone()
  if u!=None:
    # Prüfen, ob die Seite schon durchforstet wurde.
    v=self.con.execute(
    'select * from wordlocation where urlid=%d' % u[0]).fetchone()
    if v!=None: return True
  return False
```

Nun können Sie den Crawler erneut ausführen und ihn auch tatsächlich die Seiten indexieren lassen, auf die er stößt. Machen Sie das in Ihrer interaktiven Session:

```
>> reload(searchengine)
>> crawler=searchengine.crawler('searchindex.db')
>> pages= \
.. ['http://kiwitobes.com/wiki/Categorical_list_of_programming_languages.html']
>> crawler.crawl(pages)
```

Der Crawler wird vermutlich eine lange Zeit laufen. Anstatt darauf zu warten, dass er fertig wird, empfehle ich Ihnen, eine schon gefüllte Kopie der *searchindex.db* von *http://kiwitobes.com/db/searchindex.db* herunterzuladen und sie im gleichen Verzeichnis wie Ihren Python-Code zu speichern.

Wenn Sie sicherstellen wollen, dass das Crawling korrekt abgelaufen ist, können Sie versuchen, die Einträge für ein Wort zu prüfen, indem Sie die Datenbank abfragen:

```
>> [row for row in crawler.con.execute(
.. 'select rowid from wordlocation where wordid=1')]
[(1,), (46,), (330,), (232,), (406,), (271,), (192,),...
```

Die zurückgegebene Liste enthält alle URL-IDs, die »word« enthalten, was bedeutet, dass Sie erfolgreich eine Volltextsuche durchgeführt haben. Das ist ein toller Anfang, funktioniert aber immer nur mit einem Wort gleichzeitig und liefert die Dokumente auch nur in der Reihenfolge zurück, in denen sie geladen wurden. Der nächste Abschnitt wird Ihnen zeigen, wie Sie diese Funktionalität erweitern können, indem Sie die Suchen mit mehreren Wörtern gleichzeitig durchführen.

e-bol.net

Abfragen

Sie haben jetzt einen funktionierenden Crawler und eine umfassende Sammlung von indexierten Dokumenten. Sie können nun den Suchteil der Such-Engine einrichten. Zuerst erstellen Sie eine neue Klasse in *searchengine.py*, die Sie zum Suchen nutzen werden:

```
class searcher:
  def __init__(self,dbname):
    self.con=sqlite.connect(dbname)

  def __del__(self):
    self.con.close( )
```

Die Tabelle `wordlocation` bietet eine einfache Möglichkeit, Wörter mit Seiten zu verknüpfen, daher ist es recht einfach herauszufinden, welche Seiten ein bestimmtes Wort enthalten. Allerdings ist eine Such-Engine ziemlich eingeschränkt, solange sie keine Suchen mit mehreren Wörtern zulässt. Dazu brauchen Sie eine Abfragefunktion, die einen Query-String erwartet, diesen in die verschiedenen Wörter aufteilt und eine SQL-Abfrage aufbaut, mit der nur die URLs gefunden werden, die alle Wörter enthalten. Fügen Sie diese Funktion der Definition der Klasse `searcher` hinzu:

```
  def getmatchrows(self,q):
    # Strings, um die Abfrage aufzubauen.
    fieldlist='w0.urlid'
    tablelist=''
    clauselist=''
    wordids=[]

    # Die Wörter sind durch Leerzeichen getrennt.
    words=q.split(' ')
    tablenumber=0

    for word in words:
      # Wort-IDs holen.
      wordrow=self.con.execute(
        "select rowid from wordlist where word='%s'" % word).fetchone( )
      if wordrow!=None:
        wordid=wordrow[0]
        wordids.append(wordid)
        if tablenumber>0:
          tablelist+=','
          clauselist+=' and '
          clauselist+='w%d.urlid=w%d.urlid and ' % (tablenumber-1,tablenumber)
        fieldlist+=',w%d.location' % tablenumber
        tablelist+='wordlocation w%d' % tablenumber
        clauselist+='w%d.wordid=%d' % (tablenumber,wordid)
        tablenumber+=1
```

e-bol.net

```
        # Abfrage aus den Einzelteilen zusammenbauen
        fullquery='select %s from %s where %s' % (fieldlist,tablelist,clauselist)
        cur=self.con.execute(fullquery)
        rows=[row for row in cur]

        return rows,wordids
```

Diese Funktion sieht ein bisschen kompliziert aus. Es wird aber lediglich für jedes Wort eine Referenz zur Tabelle `wordlocation` erstellt, und diese IDs werden dann mit den URL-IDs verknüpft (Abbildung 4-2).

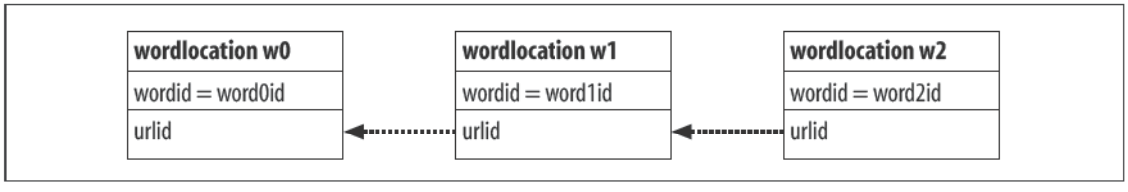

Abbildung 4-2: Tabellen-Joins für getmatchrows

So wird eine Abfrage für die beiden Wörter mit den IDs 10 und 17 aufgebaut:

```
select w0.urlid,w0.location,w1.location
from   wordlocation w0,wordlocation w1
where  w0.urlid=w1.urlid
and    w0.wordid=10
and    w1.wordid=17
```

Versuchen Sie einmal, diese Funktion für Ihre erste Mehrwortsuche aufzurufen:

```
>> reload(searchengine)
>> e=searchengine.searcher('searchindex.db')
>> e.getmatchrows('functional programming')
([(1, 327, 23), (1, 327, 162), (1, 327, 243), (1, 327, 261),
  (1, 327, 269), (1, 327, 436), (1, 327, 953),..
```

Sie werden feststellen, dass jede URL-ID mehrfach mit den verschiedenen Kombinationen der Wortpositionen zurückgegeben wird. Die nächsten Abschnitte werden Wege behandeln, die Ergebnisse zu bewerten. Bei der *contentbasierten Bewertung* werden verschiedene mögliche Metriken genutzt, die nur auf den Inhalt der Seite zurückgreifen, um die Relevanz bei der Abfrage zu bestimmen. Die *Inbound-Link-Bewertung* nutzt die Linkstruktur der Site, um zu bestimmen, was wichtig ist. Wir werden zudem eine Möglichkeit aufzeigen zu verfolgen, welche Ergebnisse die Nutzer nach der Suche tatsächlich anklicken, um die Bewertung der Ergebnisse mit der Zeit zu verfeinern.

Contentbasierte Bewertung

Sie haben es bereits geschafft, die Seiten zu erfragen, die zur Abfrage passen, aber die Reihenfolge, in der diese zurückgegeben werden, ist schlicht die Reihenfolge, in der sie durchsucht wurden. Bei einer großen Zahl von Ergebnissen wären Sie aus-

e-bol.net

giebig damit beschäftigt, eine Menge uninteressanten Krams zu durchstöbern, der zufällig die passenden Begriffe enthält, um die Seiten zu finden, die wirklich relevant sind. Um das zu vermeiden, müssen Sie Möglichkeiten finden, Seiten bei einer Abfrage einzustufen und die mit dem höchsten Wert zuerst zurückzugeben.

Dieser Abschnitt wird verschiedene Wege dazu vorstellen, eine Einstufung nur in Abhängigkeit von der Abfrage und dem Inhalt der Seite zu erreichen. Zu den Bewertungsmetriken gehören:

Wortfrequenz
: Die Häufigkeit, mit der die Wörter aus der Abfrage im Dokument vorkommen, kann dabei helfen zu entscheiden, wie relevant das Dokument ist.

Position im Dokument
: Das Hauptthema eines Dokuments wird sehr wahrscheinlich am Anfang des Dokuments auftauchen.

Wortabstand
: Wenn mehrere Wörter in der Abfrage vorkommen, sollten sie im Dokument nahe beieinanderliegen.

Die ersten Such-Engines nutzten häufig nur diese Arten von Metriken und lieferten recht brauchbare Ergebnisse zurück. Spätere Abschnitte werden noch Möglichkeiten vorstellen, die Ergebnisse mit Informationen zu verbessern, die sich außerhalb der Seiten befinden, wie zum Beispiel die Anzahl und Qualität der auf die Seite verweisenden Links.

Zuerst brauchen Sie nun eine neue Methode, die eine Abfrage erwartet, die Zeilen holt, sie in ein Dictionary steckt und in einer formatierten Liste zurückgibt. Fügen Sie diese Funktionen Ihrer Klasse searcher hinzu:

```
def getscoredlist(self,rows,wordids):
  totalscores=dict([(row[0],0) for row in rows])

  # Hier kommen später die Bewertungsfunktionen hin.
  weights=[]

  for (weight,scores) in weights:
    for url in totalscores:
      totalscores[url]+=weight*scores[url]

  return totalscores

def geturlname(self,id):
  return self.con.execute(
  "select url from urllist where rowid=%d" % id).fetchone( )[0]

def query(self,q):
  rows,wordids=self.getmatchrows(q)
  scores=self.getscoredlist(rows,wordids)
  rankedscores=sorted([(score,url) for (url,score) in scores.items( )],reverse=1)
```

e-bol.net

```
    for (score,urlid) in rankedscores[0:10]:
      print '%f\t%s' % (score,self.geturlname(urlid))
```

Bis jetzt wendet die Abfragemethode noch keine Bewertungsfunktionen auf die Ergebnisse an, gibt die URLs aber zusammen mit einem Platzhalter für die Einstufung aus:

```
>> reload(searchengine)
>> e=searchengine.searcher('searchindex.db')
>> e.query('functional programming')
0.000000 http://kiwitobes.com/wiki/XSLT.html
0.000000 http://kiwitobes.com/wiki/XQuery.html
0.000000 http://kiwitobes.com/wiki/Unified_Modeling_Language.html
...
```

Die wichtige Funktion ist hier `getscoredlist`, die Sie in diesem Abschnitt mit Leben füllen werden. Wenn Sie die Bewertungsfunktionen hinzufügen, können Sie die Aufrufe der Liste `weights` (die fett gedruckte Zeile weiter oben) hinzufügen und damit beginnen, ein paar echte Bewertungen zu erhalten.

Normalisierungsfunktion

Alle Bewertungsmethoden, die hier vorgestellt werden, liefern Dictionaries mit den URL-IDs und eine numerische Bewertung zurück. Um die Sache noch komplizierter zu machen, ist manchmal eine hohe Bewertung besser und manchmal eine niedrigere. Damit Sie die Ergebnisse aus den verschiedenen Methoden vergleichen können, brauchen Sie einen Weg, sie zu *normalisieren* – also alle in denselben Bereich und in die gleiche Richtung zu bekommen.

Die Normalisierungsfunktion wird ein Directory mit IDs und Bewertungen erhalten und ein neues Directory mit den gleichen IDs, aber Bewertungen zwischen 0 und 1 zurückgeben. Jede Bewertung wird daran gemessen, wie nahe sie am besten Ergebnis liegt, das immer eine Bewertung von 1 haben wird. Sie müssen der Funktion nur eine Liste mit Bewertungen übergeben und festlegen, ob ein niedrigerer oder ein höherer Wert besser ist:

```
def normalizescores(self,scores,smallIsBetter=0):
  vsmall=0.00001 # Fehler "Divison durch Null" vermeiden
  if smallIsBetter:
    minscore=min(scores.values())
    return dict([(u,float(minscore)/max(vsmall,l)) for (u,l) \
      in scores.items()])
  else:
    maxscore=max(scores.values())
    if maxscore==0: maxscore=vsmall
    return dict([(u,float(c)/maxscore) for (u,c) in scores.items()])
```

Jede der Bewertungsfunktionen ruft diese Funktion auf, um ihre Ergebnisse zu normalisieren und einen Wert zwischen 0 und 1 zurückzugeben.

e-bol.net

Wortfrequenz

Die Wortfrequenzmetrik bewertet eine Seite in Abhängigkeit davon, wie oft die Wörter in der Abfrage auf dieser Seite vorkommen. Wenn ich nach »python« suche, würde ich eher eine Seite über Python (oder Pythons) erhalten wollen, auf der das Wort häufig erwähnt wird, als eine Seite über einen Musiker, auf der zufällig am Ende erwähnt wird, dass er als Haustier eine Python besitzt.

Die Wortfrequenzfunktion sieht folgendermaßen aus, Sie können sie Ihrer Klasse `searcher` hinzufügen:

```
def frequencyscore(self,rows):
  counts=dict([(row[0],0) for row in rows])
  for row in rows: counts[row[0]]+=1
  return self.normalizescores(counts)
```

Diese Funktion erstellt ein Dictionary mit einem Eintrag für jede eindeutige URL-ID in *rows* und zählt, wie oft jedes Element vorkommt. Dann normalisiert sie die Bewertungen (größer ist in diesem Fall besser) und liefert das Ergebnis zurück.

Um die Frequenzbewertung für Ihre Ergebnisse zu aktivieren, ändern Sie die `weights`-Zeile in `getscoredlist` so ab:

```
  weights=[(1.0,self.frequencyscore(rows))]
```

Nun können Sie eine neue Suche ausprobieren und schauen, wie gut diese Funktion als Bewertungsfunktion arbeitet:

```
>> reload(searchengine)
>> e=searchengine.searcher('searchindex.db')
>> e.query('functional programming')
1.000000 http://kiwitobes.com/wiki/Functional_programming.html
0.262476 http://kiwitobes.com/wiki/Categorical_list_of_programming_languages.html
0.062310 http://kiwitobes.com/wiki/Programming_language.html
0.043976 http://kiwitobes.com/wiki/Lisp_programming_language.html
0.036394 http://kiwitobes.com/wiki/Programming_paradigm.html
...
```

Dies liefert die Seite zur funktionalen Programmierung an erster Stelle zurück, gefolgt von verschiedenen anderen passenden Seiten. Beachten Sie, dass »Functional Programming« viermal besser bewertet wird als das Ergebnis direkt danach. Die meisten Such-Engines geben die Bewertungen nicht an die Endanwender weiter, aber sie können in manchen Anwendungen sehr hilfreich sein. So wollen Sie vielleicht den Benutzer direkt zur besten Seite weiterleiten, wenn diese einen bestimmten Wert überschreitet, oder Ergebnisse in einer Schriftgröße proportional zur Relevanz des Ergebnisses ausgeben.

Position im Dokument

Eine andere einfache Metrik zum Bestimmen der Relevanz einer Seite in einer Abfrage ist die Position des Suchbegriffs auf der Seite. Normalerweise wird ein

e-bol.net

Suchbegriff auf einer Seite in der Nähe des Anfangs erscheinen, wenn die Seite dafür wichtig ist, vielleicht sogar im Titel. Um daraus einen Vorteil zu ziehen, kann die Such-Engine Ergebnisse besser bewerten, wenn der Suchbegriff weiter oben im Dokument erscheint. Glücklicherweise wurden die Positionen der Wörter dokumentiert, als wir die Seiten vorher indexiert haben, und der Titel der Seite ist der erste in der Liste.

Fügen Sie `searcher` diese Methode hinzu:

```
def locationscore(self,rows):
  locations=dict([(row[0],1000000) for row in rows])
  for row in rows:
    loc=sum(row[1:])
    if loc<locations[row[0]]: locations[row[0]]=loc

  return self.normalizescores(locations,smallIsBetter=1)
```

Denken Sie daran, dass das erste Element in jeder `row` die URL-ID ist, gefolgt von den Positionen aller verschiedenen Suchbegriffe. Jede ID kann mehrfach vorkommen, einmal für jede Positionskombination. Für jede Zeile summiert die Methode die Positionen aller Wörter auf und bestimmt, wie dieses Ergebnis im Vergleich zum bisher besten Ergebnis für diese URL steht. Dann werden die besten Ergebnisse an die Funktion `normalize` übergeben. Beachten Sie, dass `smallIsBetter` bedeutet, dass die URL mit der niedrigsten Position einen Wert von 1,0 erhält.

Um zu sehen, wie die Ergebnisse aussehen, wenn man nur die Bewertung über die Position nutzt, ändern Sie die `weights`-Zeile so ab:

```
weights=[(1.0,self.locationscore(rows))]
```

Nun stellen Sie die Anfrage nochmals in Ihrem Interpreter:

```
>> reload(searchengine)
>> e=searchengine.searcher('searchindex.db')
>> e.query('functional programming')
```

Sie werden feststellen, dass »Functional Programming« immer noch der Sieger ist, die anderen vorne liegenden Ergebnisse aber nun Beispiele für funktionale Programmiersprachen sind. Die vorige Suche lieferte Ergebnisse zurück, in denen die Wörter mehrfach erwähnt wurden, was die Behandlung von Programmiersprachen im Allgemeinen bevorzugte. Bei dieser Suche dagegen sorgte das Vorhandensein der Wörter im ersten Satz (zum Beispiel »Haskell is a standardized pure functional programming language«) für eine deutlich höhere Bewertung.

Es ist wichtig, sich darüber im Klaren zu sein, dass keine der bisher gezeigten Metriken in jedem Fall besser ist als die anderen. Beide Listen sind je nach Intention des Suchenden sinnvoll, und man braucht verschiedene Kombinationen der Gewichtungen, um die besten Suchergebnisse für eine bestimmte Kombination von Dokumenten und Anwendungen zu erhalten. Sie können versuchen, mit verschiedenen

e-bol.net

Gewichtungen für die beiden Metriken zu experimentieren, indem Sie Ihre `weights`-Zeile modifizieren, zum Beispiel so:

```
weights=[(1.0,self.frequencyscore(rows)),
         (1.5,self.locationscore(rows))]
```

Spielen Sie mit verschiedenen Gewichtungen und Abfragen und beobachten Sie, wie die Ergebnisse davon beeinflusst werden.

Die Verwendung der Position ist schwieriger auszutricksen als die Wortfrequenz, da Seitenautoren nur ein Wort an den Anfang eines Dokuments stellen können und eine Wiederholung keinen Einfluss auf das Ergebnis hat.

Wortabstand

Wenn eine Abfrage mehrere Wörter enthält, ist es häufig sinnvoll, Ergebnisse nach vorne zu holen, in denen sich die Wörter in der Abfrage auf der Seite nahe beieinander befinden. In den meisten Fällen sind die Nutzer beim Stellen von Mehrwortabfragen an Seiten interessiert, die die verschiedenen Wörter auch konzeptionell zusammenbringen. Das ist ein wenig flexibler als die Suchen nach festen Wortkombinationen in Anführungszeichen, die von den meisten Such-Engines unterstützt werden – dabei müssen die Wörter in der korrekten Reihenfolge ohne weitere Wörter dazwischen vorkommen. In dem hier besprochenen Fall wird die Metrik eine unterschiedliche Reihenfolge und zusätzliche Wörter zwischen den abgefragten Wörtern akzeptieren.

Die Funktion `distancescore` sieht der Funktion `locationscore` sehr ähnlich:

```
def distancescore(self,rows):
  # Wenn es nur ein Wort gibt, gewinnt jeder!
  if len(rows[0])<=2: return dict([(row[0],1.0) for row in rows])

  # Das Dictionary mit großen Werten vorbelegen.
  mindistance=dict([(row[0],1000000) for row in rows])

  for row in rows:
    dist=sum([abs(row[i]-row[i-1]) for i in range(2,len(row))])
    if dist<mindistance[row[0]]: mindistance[row[0]]=dist
  return self.normalizescores(mindistance,smallIsBetter=1)
```

Der Hauptunterschied liegt hier darin, dass die Funktion beim Abarbeiten der Positionen (in der fett gedruckten Zeile) den Abstand von jeder Position zu der vorigen Position bestimmt. Da jede Kombination aus Abständen von der Abfrage zurückgegeben wird, ist garantiert, dass der kleinste Gesamtabstand gefunden wird.

Sie können die Wortabstandsmetrik für sich allein ausprobieren, aber sie funktioniert besser, wenn sie mit anderen Metriken kombiniert wird. Versuchen Sie, `distancescore` der Liste `weights` hinzuzufügen, und passen Sie die Gewichtungen an, um zu sehen, wie sie sich auf die Ergebnisse der verschiedenen Abfragen auswirken.

e-bol.net

Eingehende Links verwenden

Die bisher besprochenen Bewertungsmetriken basierten alle auf dem Inhalt der Seite. Auch wenn viele Such-Engines immer noch so funktionieren, können die Ergebnisse häufig dadurch verbessert werden, dass man auch Informationen berücksichtigt, die andere über die Seite bereitgestellt haben, beispielsweise wer die Seite verlinkt hat und was derjenige darüber schreibt. Das ist insbesondere dann nützlich, wenn man Seiten mit dubiosem Inhalt indexiert oder solche, die eventuell von Spammern erstellt wurden, da diese sehr wahrscheinlich weniger verlinkt werden als solche mit echtem Inhalt.

Der Crawler, den Sie zu Beginn des Kapitels gebaut haben, erfasst schon alle wichtigen Informationen über die Links, daher muss er nicht angepasst werden. Die Tabelle `links` enthält die URL-IDs für Quelle und Ziel jedes Links, dem der Crawler über den Weg gelaufen ist, und die Tabelle `linkwords` verbindet die Wörter mit den Links.

Einfaches Zählen

Das Einfachste, was man mit Inbound-Links machen kann, ist, sie auf jeder Seite zu zählen und die Gesamtanzahl der Links als Metrik für die Seite zu nutzen. Akademische Artikel werden häufig so bewertet, da ihre Wichtigkeit mit der Anzahl der anderen Artikel verbunden ist, die auf sie verweisen. Die unten stehende Bewertungsfunktion erzeugt ein Dictionary mit Zählern, indem sie die Link-Tabelle für jede einzelne URL-ID in `rows` abfragt und dann die normalisierten Bewertungen zurückgibt:

```
def inboundlinkscore(self,rows):
  uniqueurls=set([row[0] for row in rows])
  inboundcount=dict([(u,self.con.execute( \
    'select count(*) from link where toid=%d' % u).fetchone( )[0]) \
      for u in uniqueurls])
  return self.normalizescores(inboundcount)
```

Natürlich wird die Verwendung lediglich nur dieser Metrik einfach alle Seiten zurückgeben, die die Suchbegriffe enthalten, geordnet nach der Anzahl der eingehenden Links, die sie haben. Im Datenbestand hat »Programming Language« viel mehr eingehende Links als »Python«, aber Sie werden vermutlich »Python« gern an erster Stelle sehen, wenn Sie nach diesem Begriff gesucht haben. Um die Relevanz mit der Bewertung zu kombinieren, müssen Sie die Inbound-Link-Metrik mit einer der weiter oben gezeigten Metriken kombinieren.

Dieser Algorithmus wichtet auch jeden eingehenden Link gleich, was zwar hübsch und fair ist, aber auch anfällig für Manipulationen, da jemand leicht viele Sites einrichten kann, die auf eine Seite zeigen, deren Bewertung erhöht werden soll. Es ist auch möglich, dass die Leute mehr an Ergebnissen interessiert sind, die die Auf-

e-bol.net

merksamkeit sehr beliebter Sites auf sich gezogen haben. Als Nächstes werden Sie erfahren, wie Sie Links von beliebten Sites beim Berechnen von Bewertungen mehr Gewicht verleihen.

Der PageRank-Algorithmus

Der PageRank-Algorithmus wurde von den Gründern von Google entwickelt, und Abwandlungen der Idee werden nun von allen großen Suchmaschinen genutzt. Dieser Algorithmus weist jeder Seite einen Wert zu, der angibt, wie wichtig die Seite ist. Die Wichtigkeit der Seite wird wiederum bestimmt durch die Wichtigkeit aller anderen Seiten, die auf sie verweisen, und durch die Anzahl der Links, die jede der anderen Seiten enthält.

Theoretisch berechnet PageRank (benannt nach Larry Page, einem seiner Erfinder) die Wahrscheinlichkeit, dass jemand, der zufällig auf Links klickt, eine bestimmte Seite erreichen wird. Je mehr eingehende Links die Seite auf anderen beliebten Seiten besitzt, desto wahrscheinlicher ist es, dass jemand dort aus purem Zufall landen wird. Wenn der Benutzer natürlich immer weiterklickt, wird er schließlich jede Seite einmal erreicht haben, aber die meisten Menschen hören mit dem Surfen nach einer Weile auf. Um das zu berücksichtigen, nutzt PageRank zudem einen *Dämpfungsfaktor* von 0,85, der angibt, dass ein Benutzer mit einer Chance von 85% auf die Links einer Seite klicken wird.

Abbildung 4-3 zeigt ein Beispiel mit Seiten und Links.

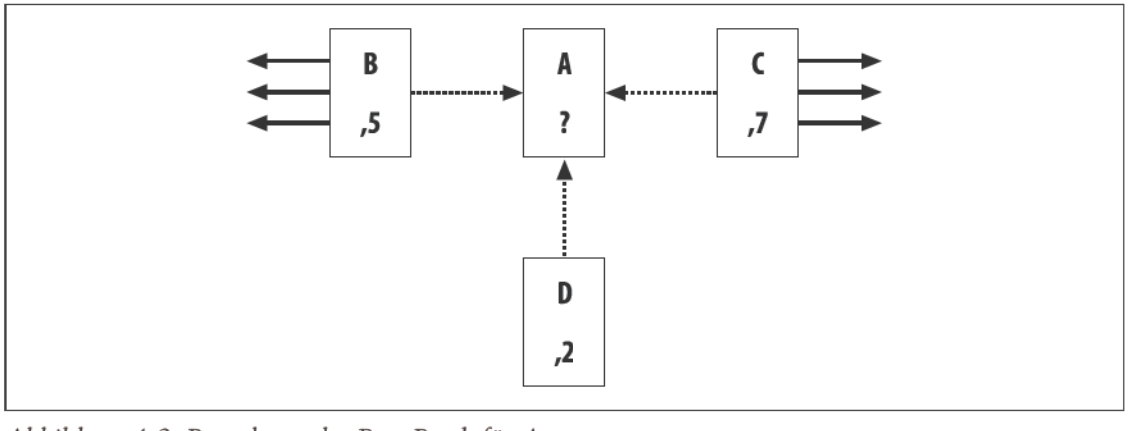

Abbildung 4-3: Berechnen des PageRank für A

Die Seiten B, C und D verweisen alle auf A und haben auch alle schon einen berechneten PageRank. B verweist dazu noch auf drei und C auf vier weitere Seiten. D linkt nur auf A. Um den PageRank von A zu erhalten, nehmen Sie den PageRank (PR) jeder der Seiten, die auf A verweisen, teilen ihn durch die Gesamtzahl der

e-bol.net

Links auf dieser Seite, multiplizieren ihn mit dem Dämpfungsfaktor von 0,85 und fügen einen Minimalwert von 0,15 hinzu. Die Berechnung für PR(A) ist:

```
PR(A) = 0.15 + 0.85 * ( PR(B)/links(B) + PR(C)/links(C) + PR(D)/links(D) )
      = 0.15 + 0.85 * ( 0.5/4 + 0.7/5 + 0.2/1 )
      = 0.15 + 0.85 * ( 0.125 + 0.14 + 0.2)
      = 0.15 + 0.85 * 0.465
      = 0.54525
```

Sie werden feststellen, dass D mehr zum PageRank von A beiträgt als B oder C, obwohl es selbst einen geringeren PageRank besitzt, weil es ausschließlich auf A verlinkt und seine gesamte Bewertung einbringen kann.

Schon nett, oder? Nun, es gibt einen kleinen Haken an der Sache – in diesem Beispiel haben alle Seiten, die auf A verlinken, schon PageRanks. Sie können den Wert einer Seite nicht berechnen, solange Sie nicht die Werte aller Seiten kennen, die auf sie verweisen, und Sie können deren Werte nicht berechnen, ohne die Werte für alle Seiten zu kennen, die auf *diese* verweisen. Wie ist es dann möglich, PageRanks für einen ganzen Satz an Seiten zu berechnen, die noch keine PageRanks haben?

Die Lösung ist, alle PageRanks auf einen bestimmten initialen Wert zu setzen (der Code wird 1,0 verwenden, aber der tatsächlich gewählte Wert ist nicht entscheidend) und die Berechnung in mehreren Iterationen durchzuführen. Nach jeder Iteration kommt der PageRank für jede Seite näher an seinen wahren Wert heran. Die Anzahl der benötigten Iterationen hängt von der Anzahl der Seiten ab, aber in dem kleinen Datenbestand, mit dem Sie arbeiten, sollte 20 ausreichend sein.

Da der PageRank sehr zeitaufwendig zu berechnen ist und auch unabhängig von der gestellten Abfrage gleich bleibt, werden Sie eine Funktion erstellen, die den PageRank im Voraus für jede URL berechnet und ihn in einer Tabelle speichert. Diese Funktion wird bei jedem Lauf alle PageRanks berechnen. Fügen Sie sie der Klasse crawler hinzu:

```
def calculatepagerank(self,iterations=20):
  # Die aktuelle PageRank-Tabelle leeren.
  self.con.execute('drop table if exists pagerank')
  self.con.execute('create table pagerank(urlid primary key,score)')

  # Jede URL mit einem PageRank von 1 initialisieren.
  self.con.execute('insert into pagerank select rowid, 1.0 from urllist')
  self.dbcommit( )

  for i in range(iterations):
    print "Iteration %d" % (i)
    for (urlid,) in self.con.execute('select rowid from urllist'):
      pr=0.15

      # Über alle Seiten laufen lassen, die diese verlinken.
      for (linker,) in self.con.execute(
      'select distinct fromid from link where toid=%d' % urlid):
```

e-bol.net

```
        # Den PageRank des Linkenden holen.
        linkingpr=self.con.execute(
        'select score from pagerank where urlid=%d' % linker).fetchone()[0]

        # Die Gesamtzahl an Links auf der verlinkenden Seite holen.
        linkingcount=self.con.execute(
        'select count(*) from link where fromid=%d' % linker).fetchone()[0]
        pr+=0.85*(linkingpr/linkingcount)
      self.con.execute(
      'update pagerank set score=%f where urlid=%d' % (pr,urlid))
    self.dbcommit()
```

Diese Funktion setzt zunächst den PageRank jeder Seite auf 1,0. Dann durchläuft sie jede URL und ermittelt den PageRank und die Gesamtzahl an Links für jeden eingehenden Link. Die fett gedruckte Seite zeigt die Formel, die für jeden der Inbound-Links angewendet wird.

Es wird ein paar Minuten dauern, diese Funktion auszuführen, aber Sie müssen sie nur laufen lassen, wenn Sie den Index aktualisieren.

```
>> reload(searchengine)
>> crawler=searchengine.crawler('searchindex.db')
>> crawler.calculatepagerank()
Iteration 0
Iteration 1
...
```

Sollten Sie neugierig sein, welche Seiten aus dem Beispiel-Datenbestand die höchsten PageRanks haben, können Sie die Datenbank direkt abfragen:

```
>> cur=crawler.con.execute('select * from pagerank order by score desc')
>> for i in range(3): print cur.next()
(438, 2.5285160000000002)
(2, 1.1614640000000001)
(543, 1.064252)
>> e.geturlname(438)
u'http://kiwitobes.com/wiki/Main_Page.html'
```

»Main Page« hat den höchsten PageRank, was nicht überrascht, da jede andere Seite bei Wikipedia auf sie verweist. Da Sie jetzt eine Tabelle mit PageRank-Werten haben, muss man für deren Nutzung nur noch eine Funktion erstellen, die die Werte aus der Datenbank ausliest und sie normalisiert. Fügen Sie diese Methode der Klasse searcher hinzu:

```
def pagerankscore(self,rows):
  pageranks=dict([(row[0],self.con.execute('select score from pagerank where
                      urlid=%d' % row[0]).fetchone()[0]) for row in rows])
  maxrank=max(pageranks.values())
  normalizedscores=dict([(u,float(l)/maxrank) for (u,l) in pageranks.items()])
  return normalizedscores
```

Erneut sollten Sie die Liste `weights` anpassen und PageRank mit aufnehmen. Versuchen Sie zum Beispiel einmal Folgendes:

e-bol.net

```
        weights=[(1.0,self.locationscore(rows)),
                 (1.0,self.frequencyscore(rows)),
                 (1.0,self.pagerankscore(rows))]
```

Die Ergebnisse Ihrer Suchen werden nun den Inhalt der Seiten und die Bewertungen mit einbeziehen. Die Abfrage für »Functional Programming« sieht nun noch besser aus:

```
2.318146 http://kiwitobes.com/wiki/Functional_programming.html
1.074506 http://kiwitobes.com/wiki/Programming_language.html
0.517633 http://kiwitobes.com/wiki/Categorical_list_of_programming_languages.html
0.439568 http://kiwitobes.com/wiki/Programming_paradigm.html
0.426817 http://kiwitobes.com/wiki/Lisp_programming_language.html
```

Der Vorteil des PageRank ist für diese wenigen gut kontrollierbaren Dokumente nicht so offensichtlich, da sie vermutlich weniger nutzlose Seiten oder Seiten, die nur dazu dienen, Aufmerksamkeit auf sich zu ziehen, enthalten, als dies normalerweise im Web der Fall ist. Aber selbst hier wird klar, dass PageRank eine nützliche Metrik ist, um hochwertige Übersichtsseiten zurückzugeben.

Verwendung des Link-Texts

Ein anderer sehr wirkungsvoller Weg, Suchergebnisse zu bewerten, ist die Auswertung der Link-Texte zu einer Seite, um zu entscheiden, wie relevant die Seite ist. Häufig erhalten Sie bessere Informationen daraus, was die Links zu einer Seite sagen, statt daraus, was die verlinkte Seite selbst sagt, da Site-Entwickler dazu tendieren, eine kurze Beschreibung dessen, auf das sie verweisen, als Link-Text zu nutzen.

Die Methode, die die Seiten aufgrund ihrer Link-Texte bewertet, hat ein zusätzliches Argument, nämlich die Liste der Wort-IDs, die gesammelt werden, wenn Sie eine Abfrage durchführen. Sie können `searcher` diese Methode hinzufügen:

```
def linktextscore(self,rows,wordids):
  linkscores=dict([(row[0],0) for row in rows])
  for wordid in wordids:
    cur=self.con.execute('select link.fromid,link.toid from linkwords,link where
                          wordid=%d and linkwords.linkid=link.rowid' % wordid)
    for (fromid,toid) in cur:
      if toid in linkscores:
        pr=self.con.execute('select score from pagerank where urlid=%d'
                                                 % fromid).fetchone()[0]
        linkscores[toid]+=pr
  maxscore=max(linkscores.values())
  normalizedscores=dict([(u,float(l)/maxscore) for (u,l) in linkscores.items()])
  return normalizedscores
```

Dieser Code läuft über alle Wörter in `wordids` und schaut nach Links, die diese Wörter enthalten. Passt das Ziel des Links zu einem der Suchergebnisse, wird der PageRank der Ausgangsseite des Links zur Bewertung der Zielseite hinzuaddiert.

e-bol.net

Eine Seite mit vielen Links von wichtigen Seiten, die die Suchbegriffe enthalten, wird eine sehr hohe Bewertung bekommen. Viele der Seiten in den Ergebnissen werden jedoch keine Links mit dem passenden Text haben und daher eine Bewertung von 0 erhalten.

Um die Link-Text-Bewertung zu aktivieren, fügen Sie einfach das Folgende irgendwo in Ihre Liste `weights` ein:

```
(1.0,self.linktextscore(rows,wordids))
```

Es gibt keine Standardwerte für die Gewichtungen dieser Metriken, die in allen Fällen funktionieren. Selbst die großen Such-Sites ändern ihre Methoden beim Bewerten der Ergebnisse ständig. Die Metriken, die Sie nutzen werden, und die Gewichtungen, die Sie zuweisen, hängen stark von der Anwendung ab, die Sie bauen wollen.

Aus Klicks lernen

Einer der Hauptvorteile von Onlineanwendungen ist, dass sie fortlaufende Rückmeldungen in Form von Benutzerverhalten bekommen. Im Fall einer Such-Engine wird jeder Benutzer direkt Informationen dazu liefern, wie viel er mit den Ergebnissen einer bestimmten Suche anfangen konnte, indem er auf ein Ergebnis klickt und sich so dafür entscheidet, nicht die anderen Ergebnisse anzuklicken. Dieser Abschnitt wird eine Möglichkeit vorstellen aufzuzeichnen, was ein Benutzer bei den Ergebnissen einer Suche anklickt und wie diese Daten dazu genutzt werden können, die Bewertungen der Ergebnisse zu verbessern.

Dazu werden Sie ein *künstliches neuronales Netz* aufbauen, das Sie trainieren, indem Sie ihm die Wörter einer Abfrage, die Suchergebnisse und die Entscheidung des Benutzers darüber, welchen Link er angeklickt hat, übergeben. Nachdem das Netz mit vielen verschiedenen Abfragen trainiert wurde, können Sie es nutzen, um die Reihenfolge der Suchergebnisse anzupassen und damit besser widerzuspiegeln, was die Benutzer in der Vergangenheit tatsächlich angeklickt haben.

Entwurf eines Netzes zum Verfolgen von Klicks

Obwohl es viele verschiedene Arten von neuronalen Netzen gibt, bestehen sie doch alle aus einer Reihe von Knoten (den *Neuronen*) und Verbindungen zwischen ihnen. Das Netz, das Sie aufbauen werden, wird als *mehrlagiges Perzeptron*-Netz (MLP – *Multilayer Perceptron*) bezeichnet. Diese Art von Netz besteht aus mehreren Schichten von Neuronen, wobei die erste Schicht die Eingaben aufnimmt – in diesem Fall die Wörter, die vom Benutzer eingegeben wurden. Die letzte Schicht sorgt für die Ausgabe, die in diesem Beispiel eine Liste von Gewichtungen für die verschiedenen URLs ist, die wir zurückgegeben haben.

e-bol.net

Es kann mehrere Schichten in der Mitte geben, aber in diesem Beispiel wird das Netz nur eine einzelne Schicht nutzen. Diese wird als *verborgene Schicht* bezeichnet, da die Welt außerhalb des Netzes niemals direkt mit ihr kommuniziert und sie auf verschiedene Eingabekombinationen reagiert. In diesem Fall ist die Eingabekombination eine Reihe von Wörtern, daher können Sie diese Schicht auch als *Query-Schicht* betrachten. Abbildung 4-4 zeigt die Struktur des Netzes. Alle Knoten der Eingabeschicht sind mit allen Knoten der verborgenen Schicht verbunden, und alle Knoten der verborgenen Schicht sind mit allen Knoten der Ausgabeschicht verbunden.

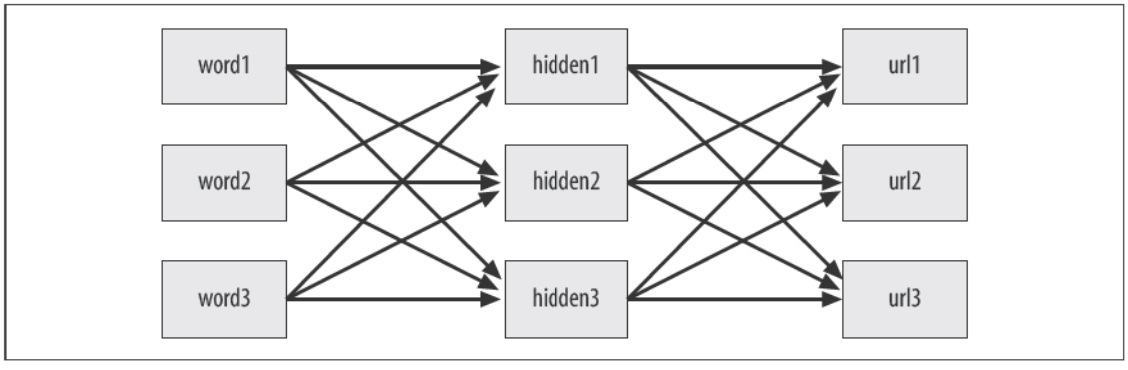

Abbildung 4-4: Design eines neuronalen Netzes zum Verfolgen von Klicks

Um das neuronale Netz nach den besten Ergebnissen einer Abfrage zu befragen, werden die Eingabeknoten für die Wörter in dieser Abfrage auf den Wert 1 gesetzt. Die Ausgabe dieser Knoten wird aktiviert, und sie versuchen, die verborgene Schicht zu aktivieren. Daraufhin werden die Knoten in der verborgenen Schicht, deren Eingangswert groß genug ist, ihre Ausgabe aktivieren und versuchen, die Knoten der Ausgabeschicht zu aktivieren.

Die Knoten der Ausgabeschicht werden dann unterschiedlich stark aktiv, und ihre Aktivitätsstufe kann genutzt werden, um zu bestimmen, wie sehr eine URL mit den Wörtern der ursprünglichen Abfrage verbunden ist. Abbildung 4-5 zeigt eine Abfrage für »world bank«. Die durchgezogenen Linien stehen für starke Verbindungen, und die fett gedruckten Texte zeigen, dass ein Knoten sehr aktiv wurde.

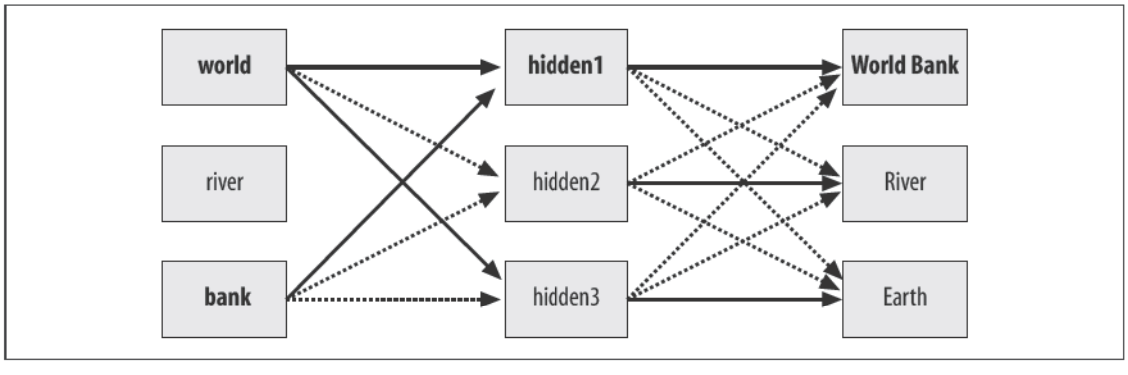

Abbildung 4-5: Antwort des neuronalen Netzes auf »world bank«

e-bol.net

Das hängt natürlich davon ab, ob die Verbindungsstärken korrekt sind. Dies wird erreicht, indem man das Netz jedes Mal *trainiert*, wenn jemand eine Suche ausführt und einen der Links im Ergebnis anklickt. Im in Abbildung 4-5 dargestellten Netz hat schon eine ganze Reihe von Leuten zuvor auf das Ergebnis »World Bank« geklickt, nachdem sie eine Suche nach »world bank« durchgeführt haben, was die Verbindungen zwischen den Wörtern und der URL verstärkt hat. Dieser Abschnitt wird Ihnen zeigen, wie das Netz mit einem Algorithmus namens `backpropagation` trainiert wird.

Sie wundern sich vielleicht, warum Sie eine solch anspruchsvolle Technik wie ein neuronales Netz brauchen, anstatt sich einfach die Abfrage zu merken und zu zählen, wie oft jedes Ergebnis angeklickt wurde. Der Vorteil des neuronalen Netzes, das Sie aufbauen werden, ist, dass es sinnvolle Vorschläge für die Ergebnisse von Abfragen machen kann, die bisher noch nicht gestellt wurden – basierend auf der Ähnlichkeit zu anderen Abfragen. Zudem sind neuronale Netze für eine ganze Reihe von Anwendungen nützlich und ergänzen Ihren Werkzeugkasten zur kollektiven Intelligenz sehr gut.

Einrichten der Datenbank

Da das neuronale Netz während der Zeit trainiert werden wird, in der die Benutzer Abfragen stellen, müssen Sie eine Repräsentation des Netzes in der Datenbank abspeichern. Die Datenbank hat bereits eine Tabelle mit Wörtern und URLs, daher benötigen Sie noch eine Tabelle für die verborgene Schicht (die `hiddennode` genannt werden wird) und zwei Tabellen mit Verbindungen (eine von der Wortschicht zur verborgenen Schicht und eine von der verborgenen Schicht zur Ausgabeschicht).

Erstellen Sie eine neue Datei namens *nn.py* und legen Sie darin eine neue Klasse mit dem Namen `searchnet` an:

```
from math import tanh
from pysqlite2 import dbapi2 as sqlite

class searchnet:
    def __init__(self,dbname):
      self.con=sqlite.connect(dbname)

    def __del__(self):
      self.con.close( )

    def maketables(self):
      self.con.execute('create table hiddennode(create_key)')
      self.con.execute('create table wordhidden(fromid,toid,strength)')
      self.con.execute('create table hiddenurl(fromid,toid,strength)')
      self.con.commit( )
```

Die Tabellen haben bisher noch keine Indizes, aber Sie können sie später hinzufügen, wenn die Geschwindigkeit zu einem Problem wird.

e-bol.net

Sie werden eine Reihe von Methoden erstellen müssen, um auf die Datenbank zuzugreifen. Die erste Methode mit dem Namen `getstrength` bestimmt die aktuelle Gewichtung einer Verbindung. Da neue Verbindungen nur bei Bedarf hinzugefügt werden, muss diese Methode einen Standardwert zurückgeben, wenn es keine Verbindung gibt. Für Verbindungen von Wörtern zur verborgenen Schicht wird der Standardwert –0,2 sein, sodass zusätzliche Wörter standardmäßig einen leicht negativen Einfluss auf die Aktivierungsstufe eines verborgenen Knotens hat. Für Links von der verborgenen Schicht zu den URLs wird die Methode einen Standardwert von 0 zurückgeben.

```
def getstrength(self,fromid,toid,layer):
  if layer==0: table='wordhidden'
  else: table='hiddenurl'
  res=self.con.execute('select strength from %s where fromid=%d and toid=%d'
                                          % (table,fromid,toid)).fetchone()
  if res==None:
      if layer==0: return -0.2
      if layer==1: return 0
  return res[0]
```

Sie werden auch eine Methode `setstrength` benötigen, um zu ermitteln, ob eine Verbindung schon besteht und um die Verbindung mit der neuen Gewichtung anzupassen oder einzurichten. Diese Methode wird von dem Code genutzt, der das Netz trainiert:

```
def setstrength(self,fromid,toid,layer,strength):
  if layer==0: table='wordhidden'
  else: table='hiddenurl'
  res=self.con.execute('select rowid from %s where fromid=%d and toid=%d'
                                        % (table,fromid,toid)).fetchone( )
  if res==None:
    self.con.execute('insert into %s (fromid,toid,strength) values (%d,%d,%f)'
                                              % (table,fromid,toid,strength))
  else:
    rowid=res[0]
    self.con.execute('update %s set strength=%f where rowid=%d'
                                      % (table,strength,rowid))
```

Meist werden beim Aufbau eines neuronalen Netzes alle Knoten im Netz im Voraus erstellt. Sie könnten ein riesiges Netz schon vorbereitet mit Tausenden von Knoten in der verborgenen Schicht und allen Verbindungen anlegen, aber in diesem Fall wird es schneller und einfacher sein, neue verborgene Knoten bei Bedarf anzulegen.

Diese Funktion wird jedes Mal einen neuen Knoten in der verborgenen Schicht anlegen, wenn eine Kombination aus Wörtern übergeben wurde, die sie noch nicht erhalten hat. Die Funktion legt dann Verbindungen mit einer Standardgewichtung zwischen den Wörtern und den verborgenen Knoten und zwischen dem Query-Knoten und den URL-Ergebnissen an, die von dieser Abfrage zurückgegeben wurden.

e-bol.net

```
def generatehiddennode(self,wordids,urls):
  if len(wordids)>3: return None
  # Prüfen, ob wir schon einen Knoten für diese Kombination aus Wörtern haben.
  createkey='_'.join(sorted([str(wi) for wi in wordids]))
  res=self.con.execute(
  "select rowid from hiddennode where create_key='%s'" % createkey).fetchone( )

  # Wenn nicht, lege ihn an.
  if res==None:
    cur=self.con.execute(
    "insert into hiddennode (create_key) values ('%s')" % createkey)
    hiddenid=cur.lastrowid
    # Standardgewichtungen eintragen.
    for wordid in wordids:
      self.setstrength(wordid,hiddenid,0,1.0/len(wordids))
    for urlid in urls:
      self.setstrength(hiddenid,urlid,1,0.1)
    self.con.commit( )
```

Im Python-Interpreter können Sie nun eine Datenbank anlegen und einen verborgenen Knoten mit ein paar Beispielwörtern und URL-IDs erstellen:

```
>> import nn
>> mynet=nn.searchnet('nn.db')
>> mynet.maketables( )
>> wWorld,wRiver,wBank =101,102,103
>> uWorldBank,uRiver,uEarth =201,202,203
>> mynet.generatehiddennode([wWorld,wBank],[uWorldBank,uRiver,uEarth])
>> for c in mynet.con.execute('select * from wordhidden'): print c
(101, 1, 0.5)
(103, 1, 0.5)
>> for c in mynet.con.execute('select * from hiddenurl'): print c
(1, 201, 0.1)
(1, 202, 0.1)
...
```

Ein neuer Knoten in der verborgenen Schicht und Verbindungen zu diesem neuen Knoten wurden nun mit Standardwerten angelegt. Die Funktion wird darauf reagieren, wenn »world« und »bank« zusammen eingegeben werden, aber diese Verbindungen werden eventuell mit der Zeit schwächer werden.

Feedforward

Nun können Sie Funktionen schreiben, die die Wörter als Eingabe nutzen, die Verbindungen im Netz aktivieren und Ausgaben für die URLs liefern.

Zuerst wählen Sie eine Funktion, die festlegt, wie jeder Knoten auf seine Eingaben reagieren sollte. Das hier beschriebene neuronale Netz wird den *Tangens Hyperbolicus* (`tanh`) nutzen, der in Abbildung 4-6 zu sehen ist.

e-bol.net

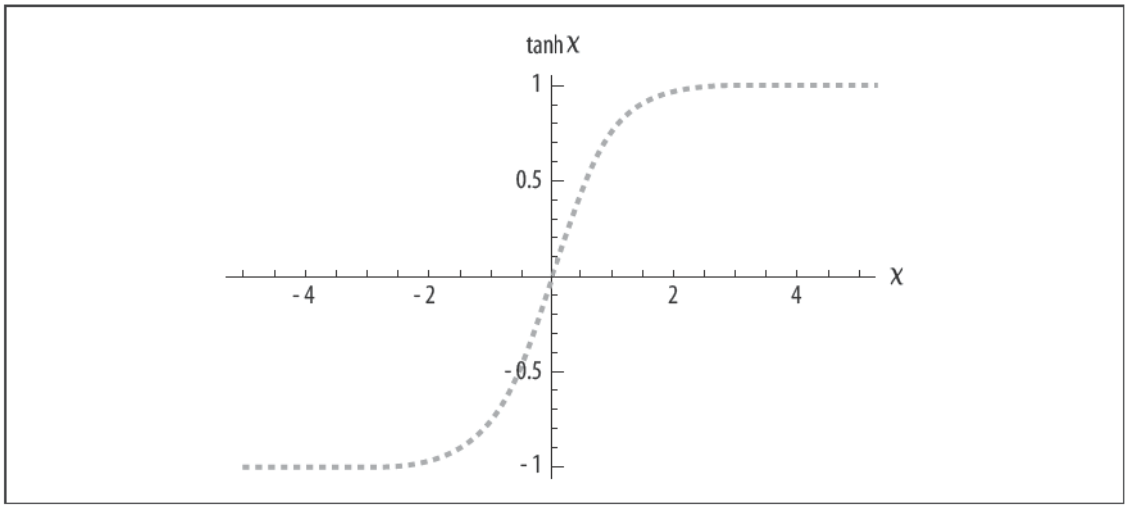

Abbildung 4-6: Die tanh-Funktion

Die x-Achse entspricht der Gesamteingabe des Knotens. Wenn sie sich 0 nähert, wird die Ausgabe schnell ansteigen. Bei einer Eingabe von 2 ist die Ausgabe schon fast bei 1 und wächst auch nicht viel weiter. Dies ist eine Form von *Sigmoid*-Funktion, die alle diese S-Form haben. Neuronale Netze nutzen so gut wie immer Sigmoid-Funktionen, um die Ausgabe der Neuronen zu bestimmen.

Bevor der `feedforward`-Algorithmus ausgeführt wird, muss die Klasse die Knoten und Verbindungen aus der Datenbank abfragen und im Speicher den Teil des Netzes aufbauen, der für eine bestimmte Abfrage relevant ist. Der erste Schritt ist dabei, eine Funktion zu erstellen, die alle Knoten in der verborgenen Schicht findet, die für eine bestimmte Abfrage wichtig sind – in diesem Fall müssen sie mit einem der Wörter in der Abfrage oder einer der URLs aus dem Ergebnis verbunden sein. Da die anderen Knoten nicht zum Bestimmen der Ausgabe und auch nicht zum Trainieren gebraucht werden, ist es nicht notwendig, sie mit aufzunehmen:

```
def getallhiddenids(self,wordids,urlids):
  l1={}
  for wordid in wordids:
    cur=self.con.execute(
    'select toid from wordhidden where fromid=%d' % wordid)
    for row in cur: l1[row[0]]=1
  for urlid in urlids:
    cur=self.con.execute(
    'select fromid from hiddenurl where toid=%d' % urlid)
    for row in cur: l1[row[0]]=1
  return l1.keys()
```

Sie brauchen außerdem eine Methode, um das relevante Netz mit allen aktuellen Gewichtungen aus der Datenbank aufzubauen. Diese Funktion definiert eine ganze Menge Instanzvariablen für diese Klasse – die Liste mit Wörtern, Abfrageknoten und URLs, den Ausgangswert jedes Knotens und die Gewichtungen jeder Verbin-

e-bol.net

dung zwischen den Knoten. Die Gewichtungen werden mithilfe der Funktion aus der Datenbank geholt, die wir weiter oben definiert haben.

```
def setupnetwork(self,wordids,urlids):
    # Werte-Liste
    self.wordids=wordids
    self.hiddenids=self.getallhiddenids(wordids,urlids)
    self.urlids=urlids

    # Knoten-Ausgaben
    self.ai = [1.0]*len(self.wordids)
    self.ah = [1.0]*len(self.hiddenids)
    self.ao = [1.0]*len(self.urlids)

    # Gewichtungs-Matrix erstellen
    self.wi = [[self.getstrength(wordid,hiddenid,0)
                for hiddenid in self.hiddenids]
               for wordid in self.wordids]
    self.wo = [[self.getstrength(hiddenid,urlid,1)
                for urlid in self.urlids]
               for hiddenid in self.hiddenids]
```

Jetzt können Sie endlich den `feedforward`-Algorithmus definieren. Dieser erwartet eine Liste mit Eingabewerten, befördert sie durch das Netz und liefert die Ausgabewerte aller Knoten in der Ausgabeschicht zurück. Da Sie in diesem Fall nur ein Netz mit Wörtern aus der Abfrage erstellt haben, wird die Ausgabe aller Eingabeknoten immer 1 sein:

```
def feedforward(self):
    # Eingaben sind nur die Abfragewörter.
    for i in range(len(self.wordids)):
        self.ai[i] = 1.0

    # Aktivieren der verborgenen Knoten.
    for j in range(len(self.hiddenids)):
        sum = 0.0
        for i in range(len(self.wordids)):
            sum = sum + self.ai[i] * self.wi[i][j]
        self.ah[j] = tanh(sum)

    # Aktivieren der Ausgabeknoten.
    for k in range(len(self.urlids)):
        sum = 0.0
        for j in range(len(self.hiddenids)):
            sum = sum + self.ah[j] * self.wo[j][k]
        self.ao[k] = tanh(sum)

    return self.ao[:]
```

Der `feedforward`-Algorithmus durchläuft alle Knoten in der verborgenen Schicht und summiert die Ausgaben aus der Eingabeschicht, multipliziert mit der Gewichtung der Verbindungen. Die Ausgabe jedes Knotens ist die `tanh`-Funktion der

e-bol.net

Summe aller seiner Eingaben, die dann an die Ausgabeschicht übergeben werden. Die Ausgabeschicht macht das Gleiche: Multiplizieren der Ausgaben der vorigen Schicht mit den Gewichtungen und Anwenden der `tanh`-Funktion, um die endgültige Ausgabe zu erhalten. Es ist leicht, das Netz um weitere Schichten zu ergänzen, indem man die Ausgabe einer Schicht als Eingabe der nächsten Schicht verwendet.

Jetzt können Sie eine kurze Funktion schreiben, die das Netz aufsetzt und `feedforward` verwendet, um die Ausgaben für eine Kombination von Wörtern und URLs zu ermitteln:

```
def getresult(self,wordids,urlids):
  self.setupnetwork(wordids,urlids)
  return self.feedforward( )
```

Sie können Python nutzen, um dies im Netz auszuprobieren:

```
>> reload(nn)
>> mynet=nn.searchnet('nn.db')
>> mynet.getresult([wWorld,wBank],[uWorldBank,uRiver,uEarth])
[0.76,0.76,0.76]
```

Die Werte in der zurückgegebenen Liste entsprechen der Relevanz der Eingabe-URLs. Wenig überraschend ist, dass das neuronale Netz die gleichen Antworten für jede URL gibt – es wurde ja noch nicht trainiert.

Training per Backpropagation

Jetzt wird es richtig interessant. Das Netz erwartet Eingaben und gibt Ausgaben zurück, aber da ihm nicht erzählt wurde, wie ein gutes Ergebnis aussehen soll, sind die Ergebnisse ziemlich nutzlos. Sie werden das Netz jetzt trainieren, indem Sie ihm ein paar echte Beispiele dazu zeigen, was die Leute gesucht haben, welche Ergebnisse zurückkamen und für welchen Link sich die Benutzer dann entschieden haben.

Dazu brauchen Sie einen Algorithmus, der die Gewichtungen der Verbindungen zwischen den Knoten so anpasst, dass sie besser wiedergeben, was dem Netz als richtige Antwort vermittelt wurde. Die Gewichtungen müssen vorsichtig angepasst werden, da Sie nicht davon ausgehen können, dass jeder Benutzer auf eine Antwort klicken wird, die für alle sinnvoll ist. Der Algorithmus, den Sie nutzen werden, heißt *Backpropagation*, da er durch das Netz zurückwandert und die Gewichtungen anpasst.

Wenn Sie ein Netz trainieren, kennen Sie immer die gewünschte Ausgabe jedes Knotens in der Ausgabeschicht. In diesem Fall sollte sie sich in Richtung 1 bewegen, wenn der Benutzer dieses Ergebnis angeklickt hat, und gegen 0, wenn er es nicht tat. Die einzige Möglichkeit, die Ausgabe eines Knotens zu ändern, ist, die Gesamteingabe dieses Knotens anzupassen.

e-bol.net

Um zu ermitteln, wie stark die Gesamteingabe geändert werden sollte, muss der Trainingsalgorithmus die Steigung der `tanh`-Funktion für den aktuellen Ausgabewert kennen. In der Mitte der Funktion, in der die Ausgabe 0,0 ist, ist die Steigung sehr steil, daher sorgt eine kleine Änderung der Eingabe für eine große Veränderung bei der Ausgabe. Nähert sich die Ausgabe –1 oder 1, hat eine Änderung der Eingabe eine geringere Auswirkung auf die Ausgabe. Die Steigung der Funktion für einen beliebigen Wert wird durch diese Funktion festgelegt, die Sie am Anfang von *nn.py* hinzufügen können:

```
def dtanh(y):
    return 1.0-y*y
```

Bevor die Backpropagation-Methode ausgeführt wird, ist es notwendig, `feedforward` laufen zu lassen, damit die momentane Ausgabe jedes Knotens in den Instanzvariablen abgelegt wird. Der Backpropagation-Algorithmus führt dann die folgenden Schritte durch.

Für jeden Knoten in der Ausgabeschicht:

1. Berechne die Differenz zwischen dem aktuellen Ausgabewert des Knotens und dem gewünschten Wert.
2. Nutze die Funktion `dtanh`, um zu berechnen, wie stark sich die Gesamteingabe des Knotens ändern muss.
3. Ändere die Gewichtung jeder eingehenden Verbindung proportional zur aktuellen Gewichtung und der Lernrate.

Für jeden Knoten in der verborgenen Schicht:

1. Ändere die Ausgabe des Knotens um die Summe der Gewichtungen jeder Verbindung zu einem Ausgabeknoten, multipliziert mit dem Änderungsfaktor, um den sich der Zielknoten ändern muss.
2. Nutze die Funktion `dtanh`, um zu bestimmen, wie stark sich die Gesamteingabe des Knotens ändern muss.
3. Ändere die Gewichtung jeder Eingabeverbindung proportional zur aktuellen Gewichtung der Verbindung und der Lernrate.

Die Implementierung dieses Algorithmus berechnet allerdings die Fehler im Voraus und passt dann die Gewichtungen an, da alle Berechnungen auf dem Wissen über die aktuellen Gewichtungen basieren und nicht auf den angepassten Gewichtungen. Hier ist der Code für den Algorithmus, den Sie der Klasse `searchnet` hinzufügen können:

```
    def backPropagate(self, targets, N=0.5):
        # Fehler für die Ausgabe berechnen.
        output_deltas = [0.0] * len(self.urlids)
        for k in range(len(self.urlids)):
            error = targets[k]-self.ao[k]
            output_deltas[k] = dtanh(self.ao[k]) * error
```

e-bol.net

```
    # Fehler für die verborgene Schicht berechnen.
    hidden_deltas = [0.0] * len(self.hiddenids)
    for j in range(len(self.hiddenids)):
        error = 0.0
        for k in range(len(self.urlids)):
            error = error + output_deltas[k]*self.wo[j][k]
        hidden_deltas[j] = dtanh(self.ah[j]) * error

    # Ausgabegewichtungen anpassen.
    for j in range(len(self.hiddenids)):
        for k in range(len(self.urlids)):
            change = output_deltas[k]*self.ah[j]
            self.wo[j][k] = self.wo[j][k] + N*change

    # Eingabegewichtungen anpassen.
    for i in range(len(self.wordids)):
        for j in range(len(self.hiddenids)):
            change = hidden_deltas[j]*self.ai[i]
            self.wi[i][j] = self.wi[i][j] + N*change
```

Nun brauchen Sie nur noch eine einfache Methode zum Einrichten des Netzes, zum Ausführen von `feedforward` und dem Durchlauf der Backpropagation. Diese Methode erhält die Listen von `wordids`, `urlids` und eine ausgewählte URL:

```
def trainquery(self,wordids,urlids,selectedurl):
  # Einen verborgenen Knoten bei Bedarf erstellen.
  self.generatehiddennode(wordids,urlids)

  self.setupnetwork(wordids,urlids)
  self.feedforward( )
  targets=[0.0]*len(urlids)
  targets[urlids.index(selectedurl)]=1.0
  error = self.backPropagate(targets)
  self.updatedatabase( )
```

Um die Ergebnisse zu speichern, brauchen Sie dazu noch eine Methode, die die Datenbank mit den neuen Gewichtungen versorgt, die in den Instanzvariablen `wi` und `wo` abgelegt sind:

```
def updatedatabase(self):
  # Auf die Datenbankwerte setzen.
  for i in range(len(self.wordids)):
      for j in range(len(self.hiddenids)):
          self.setstrength(self.wordids[i],self. hiddenids[j],0,self.wi[i][j])
  for j in range(len(self.hiddenids)):
      for k in range(len(self.urlids)):
          self.setstrength(self.hiddenids[j],self.urlids[k],1,self.wo[j][k])
  self.con.commit( )
```

Jetzt können Sie einen einfachen Test mit der vorher schon durchgeführten Abfrage starten, um zu sehen, wie das Netz auf das Training reagiert:

```
>> reload(nn)
>> mynet=nn.searchnet('nn.db')
```

e-bol.net

```
>> mynet.trainquery([wWorld,wBank],[uWorldBank,uRiver,uEarth],uWorldBank)
>> mynet.getresult([wWorld,wBank],[uWorldBank,uRiver,uEarth])
[0.335,0.055,0.055]
```

Die Ausgabe der World Bank-URL steigt und die der anderen URLs sinkt, nachdem das Netz gelernt hat, dass ein bestimmter Benutzer diese Auswahl getroffen hat. Je mehr Benutzer sich dafür entscheiden, desto größer wird der Unterschied werden.

Trainingstest

Bisher haben Sie erfahren, dass das Trainieren mit einem Beispielergebnis die Ausgabe für dieses Ergebnis erhöht. Das ist zwar nützlich, zeigt aber nicht wirklich, wozu neuronale Netze in der Lage sind – nämlich logisch mit Eingabewerten umzugehen, die sie noch nie vorher gesehen haben. Probieren Sie diesen Code in Ihrer interaktiven Python-Session ein:

```
>> allurls=[uWorldBank,uRiver,uEarth]
>> for i in range(30):
...     mynet.trainquery([wWorld,wBank],allurls,uWorldBank)
...     mynet.trainquery([wRiver,wBank],allurls,uRiver)
...     mynet.trainquery([wWorld],allurls,uEarth)
...
>> mynet.getresult([wWorld,wBank],allurls)
[0.861, 0.011, 0.016]
>> mynet.getresult([wRiver,wBank],allurls)
[-0.030, 0.883, 0.006]
>> mynet.getresult([wBank],allurls)
[0.865, 0.001, -0.85]
```

Obwohl das Netz zuvor noch keine Abfrage für »bank« vorgesetzt bekommen hat, liefert es einen sinnvollen Vorschlag zurück. Und nicht nur das, es gibt der URL der World Bank auch noch eine deutlich bessere Bewertung als der River-URL, obwohl »bank« in den Trainings-Beispielabfragen genauso häufig mit »river« wie mit World Bank verbunden worden war. Das Netz hat nicht nur gelernt, welche URLs mit welchen Abfragen verbunden sind, sondern auch, was die entscheidenden Wörter in einer bestimmten Abfrage sind – etwas, das man nicht mit einer einfachen Abfrage-URL-Korrelation hinbekommt.

Verbindung zur Such-Engine

Die Methode `query` der Klasse `searcher` erhält eine Liste mit URL-IDs und Wort-IDs mit der Aufgabe, die Ergebnisse zu erstellen und auszugeben. Sie können die Methode diese Ergebnisse zurückgeben lassen, indem Sie die folgende Zeile an das Ende von `query` in *searchengine.py* anfügen:

```
return wordids,[r[1] for r in rankedscores[0:10]]
```

Das kann direkt an die Methode `trainquery` von `searchnet` weitergereicht werden.

e-bol.net

Die Methode, mit der erfasst wird, welches der Ergebnisse der Benutzer am stärksten bevorzugte, hängt vom Design Ihrer Anwendung ab. Es ist möglich, dass eine Webseite eine zwischengeschaltete Seite enthält, die den Klick aufzeichnet und `trainquery` aufruft, bevor sie zum eigentlichen Suchergebnis weiterleitet, oder Sie könnten die Benutzer die Relevanz der Suchergebnisse selbst bewerten lassen, um auch damit Ihren Algorithmus weiter zu verbessern.

Der letzte Schritt beim Aufbau des künstlichen neuronalen Netzes ist das Erstellen einer neuen Methode in der Klasse `searcher`, um es Ihnen zu ermöglichen, Ihre Ergebnisse zu gewichten. Diese Funktion sieht den anderen Gewichtungsfunktionen sehr ähnlich. Als Erstes müssen Sie dabei die Klasse für das neuronale Netz in *searchengine.py* importieren:

```
import nn
mynet=nn.searchnet('nn.db')
```

Dann fügen Sie diese Methode der Klasse `searcher` hinzu:

```
  def nnscore(self,rows,wordids):
    # Einzelne URL-IDs als geordnete Liste erhalten.
    urlids=[urlid for urlid in set([row[0] for row in rows])]
    nnres=mynet.getresult(wordids,urlids)
    scores=dict([(urlids[i],nnres[i]) for i in range(len(urlids))])
    return self.normalizescores(scores)
```

Auch hier können Sie experimentieren, indem Sie diese Funktion in Ihre Liste `weights` mit verschiedenen Gewichtungen einfügen. In der Praxis ist es besser, mit der Aufnahme der Funktion direkt in Ihre Bewertungsgewichtungen so lange zu warten, bis das Netz mit einer großen Zahl verschiedener Beispiele trainiert wurde.

Dieses Kapitel hat einen großen Bereich an Möglichkeiten aufgezeigt, eine Such-Engine zu entwickeln, aber im Vergleich zu dem, was möglich ist, ist er noch sehr eingeschränkt. Die Übungen werden ein paar weitere Ideen aufzeigen. Dieses Kapitel hat sich nicht sehr um Performance gekümmert – wodurch eine Indexierung von Millionen von Seiten lange dauern würde –, aber was Sie aufgebaut haben, wird ausreichen, um 100.000 Seiten zu verarbeiten, also genug für eine Nachrichten-Site oder ein Firmen-Intranet.

Übungen

1. *Wörter auftrennen*. Die Methode `separatewords` geht bisher davon aus, dass jeder Nicht-Buchstabe ein Trennzeichen ist, was bedeutet, dass sie Einträge wie »C++«, »$20«, »Ph.D.« oder »617-555-1212« nicht ordentlich indexiert. Was wäre ein besserer Weg, um Wörter zu unterteilen? Kann man Leerraum als Trennzeichen nutzen? Schreiben Sie eine bessere Funktion zum Unterteilen von Wörtern.

e-bol.net

2. *Boolesche Operationen.* Viele Such-Engines unterstützen Boolesche Abfragen, bei denen der Benutzer Suchen aufbauen kann wie zum Beispiel »python OR perl«. Eine OR-Suche kann man umsetzen, indem man die Abfragen getrennt durchführt und die Ergebnisse kombiniert – aber wie ist es mit »python AND (program OR code)«? Passen Sie die Abfragemethoden an, damit diese ein paar grundlegende Boolesche Operationen unterstützen.
3. *Exakte Treffer.* Such-Engines bieten häufig Abfragen für »Exakte Treffer« an, bei denen die Wörter auf der Seite den Wörtern in der Abfrage in der gleichen Reihenfolge und ohne zusätzliche Wörter dazwischen entsprechen müssen. Erstellen Sie eine neue Version von `getrows`, die nur Ergebnisse zurückgibt, die exakte Treffer sind. (Hinweis: Sie können in ermitteln Subtraktionen verwenden, um den Abstand zwischen den Wortpositionen zu erhalten.)
4. *Suche nach langen/kurzen Dokumenten.* Manchmal ist die Länge einer Seite ein entscheidender Faktor bei der Suche durch eine Anwendung oder einen Benutzer. Ein Benutzer kann daran interessiert sein, einen langen Artikel über ein schwieriges Thema zu finden oder auch einen kurzen Überblick über ein kleines Programm. Schreiben Sie eine Gewichtungsfunktion, die in Abhängigkeit von seinen Parametern langen oder kurzen Dokumenten den Vorzug gibt.
5. *Gerechtere Wortfrequenz.* Die Metrik zum Zählen von Wörtern gibt längeren Dokumenten den Vorzug, da ein solches Dokument mehr Wörter beinhaltet und daher auch das Zielwort häufiger vertreten sein kann. Schreiben Sie eine neue Metrik, die die Frequenz als Prozentsatz der Anzahl der Wörter im Dokument ermittelt.
6. *Suche nach Inbound-Links.* Ihr Code bewertet Elemente basierend auf dem Text der eingehenden Links, aber sie müssen vorher schon durch den Inhalt der Seite gefunden worden sein. Manchmal enthält die wichtigste Seite die Suchwörter gar nicht, aber eine Menge Links mit dem Suchtext zeigen darauf – dies ist häufig bei Links auf Bilder der Fall. Passen Sie den Suchcode so an, dass er auch Ergebnisse enthält, bei denen ein eingehender Link einige der Suchbegriffe enthält.
7. *Verschiedene Trainingsoptionen.* Das neuronale Netz wird mit einer Reihe von Nullen für alle URLs trainiert, die ein Benutzer nicht angeklickt hat, und einer 1 für die URL, die er wählte. Passen Sie die Trainingsfunktion so an, dass sie stattdessen für eine Anwendung funktioniert, in der die Benutzer den Ergebnissen eine Bewertung zwischen 1 und 5 geben können.
8. *Zusätzliche Schichten.* Ihr neuronales Netz hat nur eine verborgene Schicht. Passen Sie die Klasse so an, dass sie eine beliebige Zahl an verborgenen Schichten unterstützt, was bei der Initialisierung festgelegt werden kann.

e-bol.net

KAPITEL 5

Optimierung

Dieses Kapitel wird sich damit befassen, kollaborative Probleme mit verschiedenen Techniken zu lösen, die als *stochastische Optimierung* bezeichnet werden. Optimierungstechniken werden im Allgemeinen bei Problemen genutzt, die viele mögliche Lösungen mit vielen Variablen haben und deren Ergebnisse sich massiv je nach Kombination dieser Variablen ändern können. Diese Optimierungstechniken finden in einem weit gefassten Bereich Anwendung: Wir verwenden sie in der Physik, um Molekülbewegungen zu untersuchen, in der Biologie zum Vorhersagen von Proteinstrukturen und in der Informatik zum Bestimmen der schlechtestmöglichen Laufzeit eines Algorithmus. Die NASA nutzt Optimierungstechniken sogar dazu, Antennen zu entwerfen, die die gewünschten Operationscharakteristiken haben und deren Gestalt ein menschlicher Designer so nie entworfen hätte.

Bei der Optimierung wird die beste Lösung für ein Problem dadurch gefunden, dass viele verschiedene Lösungen ausgetestet und bewertet werden, um ihre Qualität zu bestimmen. Sie wird normalerweise in Fällen genutzt, in denen es zu viele mögliche Lösungen gibt, als dass sie alle ausprobiert werden könnten. Die einfachste, aber am wenigsten effektive Methode beim Suchen nach Lösungen ist, einfach ein paar Tausend zufällige Variationen auszuprobieren und zu schauen, welche davon die beste ist. Effektivere Methoden, die wir in diesem Kapitel besprechen werden, nutzen intelligente Abwandlungen der Lösungen, bei denen man davon ausgeht, dass sie dadurch verbessert werden.

Das erste Beispiel in diesem Kapitel dreht sich um die Planung einer Gruppenreise. Jeder, der schon einmal eine Reise für eine Gruppe von Menschen geplant hat oder eventuell sogar auch nur für eine einzelne Person, weiß, dass es eine Reihe von Eingabeparametern gibt, wie zum Beispiel, wann jede Person fliegen sollte, wie viele Autos zu mieten sind und welcher Flughafen sich am ehesten anbietet. Es müssen auch viele Ausgabewerte berücksichtigt werden, wie zum Beispiel die Gesamtkosten, die Wartezeit an den Flughäfen und die benötigten Urlaubstage beim Arbeitgeber. Da die Eingabewerte nicht mit einer einfachen Formel auf die Ausgabewerte

e-bol.net

abgebildet werden können, führt das Problem, die beste Lösung zu finden, zur Optimierung.

Die anderen Beispiele im Kapitel zeigen die Flexibilität der Optimierung, indem sie zwei vollständig verschiedene Probleme beschreiben: Wie verteilt man begrenzte Ressourcen in Abhängigkeit von den Vorlieben der Teilnehmer, und wie visualisiert man ein soziales Netzwerk mit möglichst wenig sich kreuzenden Linien? Am Ende des Kapitels werden Sie dazu in der Lage sein, auch andere Arten von Problemen anzugehen, die per Optimierung gelöst werden können.

Gruppenreisen

Die Planung einer Reise für eine Gruppe von Personen (in diesem Beispiel die Familie Glass), die von verschiedenen Ausgangspunkten startend alle am gleichen Ort eintreffen sollen, ist immer eine Herausforderung und für uns auch ein interessantes Optimierungsproblem. Zunächst erstellen Sie eine neue Datei *optimization.py* und fügen den folgenden Code ein:

```
import time
import random
import math

people = [('Seymour','BOS'),
          ('Franny','DAL'),
          ('Zooey','CAK'),
          ('Walt','MIA'),
          ('Buddy','ORD'),
          ('Les','OMA')]

# LaGuardia-Flughafen in New York
destination='LGA'
```

Die Familienmitglieder sind über das ganze Land verteilt und möchten sich in New York treffen. Sie möchten alle an einem Tag ankommen und auch an einem (anderen) Tag wieder abreisen und den Weg vom und zum Flughafen gemeinsam zurücklegen. Es gibt pro Tag Dutzende von Flügen mit unterschiedlichen Abflugzeiten nach New York von jedem der Ausgangspunkte der Teilnehmer. Die Flüge haben zudem verschiedene Preise und brauchen unterschiedlich lange.

Sie können eine Beispieldatei mit Flugdaten von *http://kiwitobes.com/optimize/schedule.txt* herunterladen.

Diese Datei enthält den Abflugort, das Ziel, die Abflugs- und Ankunftszeit sowie die Preise einer ganzen Reihe von Flügen – alles durch Kommata voneinander getrennt.

```
LGA,MIA,20:27,23:42,169
MIA,LGA,19:53,22:21,173
LGA,BOS,6:39,8:09,86
BOS,LGA,6:17,8:26,89
LGA,BOS,8:23,10:28,149
```

e-bol.net

Laden Sie diese Daten in ein Dictionary mit dem Ausgangspunkt `origin` und dem Ziel `dest` als Schlüssel sowie eine Liste potenzieller Flüge als Werte. Fügen Sie diesen Code in *optimization.py* ein, um die Daten zu laden:

```
flights={}
#
for line in file('schedule.txt'):
  origin,dest,depart,arrive,price=line.strip().split(',')
  flights.setdefault((origin,dest),[])

  # Der Liste möglicher Flüge Details hinzufügen.
  flights[(origin,dest)].append((depart,arrive,int(price)))
```

Es ist hier zudem nützlich, eine Hilfsfunktion `getminutes` zu bauen, die berechnet, wie viele Minuten an einem Tag für eine gegebene Zeit schon vergangen sind. Damit ist es einfacher, Flug- und Wartezeiten zu berechnen. Fügen Sie diese Funktion der Datei *optimization.py* hinzu:

```
def getminutes(t):
  x=time.strptime(t,'%H:%M')
  return x[3]*60+x[4]
```

Die Herausforderung liegt nun darin zu entscheiden, welchen Flug jedes Familienmitglied nutzen sollte. Natürlich ist ein Ziel, den Gesamtpreis aller Flüge niedrig zu halten, aber es gibt viele weitere mögliche Faktoren, die die optimale Lösung berücksichtigen und möglichst minimieren soll, wie zum Beispiel die Gesamtwartezeit am Flughafen oder die Gesamtflugzeit. Diese anderen Faktoren werden gleich noch genauer behandelt.

Lösungen repräsentieren

Wenn man an ein Problem wie dieses herangeht, ist es notwendig festzulegen, wie eine mögliche Lösung repräsentiert werden soll. Die Optimierungsfunktionen, die Sie später kennenlernen werden, sind generisch genug, um mit vielen möglichen Problemarten zu arbeiten, daher ist es wichtig, eine einfache Repräsentation zu wählen, die nicht nur für das Gruppenreise-Problem genutzt werden kann. Eine sehr verbreitete Repräsentation ist eine Liste mit Zahlen. In diesem Fall kann jede Zahl für einen Flug stehen, den eine Person wählen wird, wobei 0 der erste Flug des Tages ist, 1 der zweite und so weiter. Da jede Person einen Hin- und einen Rückflug braucht, ist die Liste doppelt so lang wie die Anzahl an Personen.

So stellt zum Beispiel diese Liste:

```
[1,4,3,2,7,3,6,3,2,4,5,3]
```

eine Lösung dar, in der Seymour den zweiten Flug des Tages von Boston nach New York nimmt und bei der Rückreise den fünften Flug zurück nach Boston. Franny nimmt den vierten Flug von Dallas nach New York und den dritten Flug zurück.

e-bol.net

Da es schwierig sein wird, Lösungen aus der Zahlenliste auszulesen, brauchen Sie noch eine Routine, die alle Flüge, für die sich die Menschen entschieden haben, in einer hübschen Tabelle ausgibt. Fügen Sie diese Funktion der Datei *optimization.py* hinzu:

```
def printschedule(r):
  for d in range(len(r)/2):
    name=people[d][0]
    origin=people[d][1]
    out=flights[(origin,destination)][r[d]]
    ret=flights[(destination,origin)][r[d+1]]
    print '%10s%10s %5s-%5s $%3s %5s-%5s $%3s' % (name,origin,
                                                  out[0],out[1],out[2],
                                                  ret[0],ret[1],ret[2])
```

Damit werden pro Zeile der Name der Person, der Abflugsort, Abflugs- und Ankunftszeit sowie die Preise für Hin- und Rückflug angegeben. Testen Sie diese Funktion in Ihrer Python-Session:

```
>>> import optimization
>>> s=[1,4,3,2,7,3,6,3,2,4,5,3]
>>> optimization.printschedule(s)
   Seymour    Boston 12:34-15:02 $109 12:08-14:05 $142
    Franny    Dallas 12:19-15:25 $342  9:49-13:51 $229
     Zooey     Akron  9:15-12:14 $247 15:50-18:45 $243
      Walt     Miami 15:34-18:11 $326 14:08-16:09 $232
     Buddy   Chicago 14:22-16:32 $126 15:04-17:23 $189
       Les     Omaha 15:03-16:42 $135  6:19- 8:13 $239
```

Unabhängig vom Preis gibt es mit diesem Flugplan ein paar Probleme. Insbesondere, da die Familienmitglieder zusammen vom Flughafen wegfahren und auch wieder hinfahren wollen, muss jeder am Rückflugtag um 6 Uhr morgens am Flughafen sein, damit Les seinen Flieger erreicht, obwohl manche erst kurz vor 16 Uhr nachmittags abfliegen. Um die beste Kombination zu bestimmen, muss das Programm eine Möglichkeit haben, die verschiedenen Eigenschaften der Flugpläne zu gewichten, um zu entscheiden, welcher Plan der beste ist.

Die Zielfunktion

Die *Zielfunktion* (*Cost Function*, daher auch Kostenfunktion) ist der entscheidende Schlüssel, mit dem alle Probleme rund um die Optimierung gelöst werden können, und sie ist meist auch das Element, das am schwierigsten zu bestimmen ist. Das Ziel jedes Optimierungsalgorithmus ist, eine Reihe von Eingaben – hier Flüge – zu finden, die die Zielfunktion minimieren. Daher muss die Zielfunktion einen Wert zurückgeben, der angibt, wie schlecht eine Lösung ist. Es gibt keinen bestimmten Maßstab für *Schlechtheit*, wichtig ist nur, dass die Funktion größere Werte für schlechtere Lösungen zurückgibt.

e-bol.net

Häufig ist es schwierig zu bestimmen, was eine Lösung unter Berücksichtigung der vielen Variablen gut oder schlecht macht. Betrachten Sie nur ein paar der Dinge, die in dem Beispiel mit der Gruppenreise gemessen werden können:

Preis
: Der Gesamtpreis aller Flugtickets oder stattdessen ein gewichtetes Mittel, das die finanzielle Situation der einzelnen Personen berücksichtigt.

Reisezeit
: Die Gesamtreisezeit, die jeder im Flugzeug verbringen muss.

Wartezeit
: Die Zeit, die am Flughafen damit verbracht wird, auf die Ankunft der anderen Mitglieder der Gruppe zu warten.

Abflugzeit
: Flüge, die zu früh am Morgen starten, können für zusätzlichen Ärger sorgen, weil Reisende dann auf Schlaf verzichten müssen.

Auto-Mietdauer
: Wenn die Gruppe ein Auto mietet, muss sie es am Tag der Rückreise vor der Zeit zurückbringen, zu der sie es am Ankunftstag gemietet hat, oder ansonsten einen ganzen zusätzlichen Tag bezahlen.

Es ist nicht schwierig, sich noch mehr Aspekte für einen bestimmten Flugplan zu überlegen, die die ganze Angelegenheit angenehmer oder beschwerlicher machen. Jedes Mal, wenn Sie die Aufgabe haben, die beste Lösung für ein kompliziertes Problem zu finden, müssen Sie entscheiden, welche Faktoren die wichtigen sind. Auch wenn das schwierig sein kann, hat es den großen Vorteil, dass Sie nach Erledigung dieser Aufgabe die Optimierungsalgorithmen aus diesem Kapitel für so gut wie jedes Problem nutzen können und dabei nur minimale Änderungen vornehmen müssen.

Nachdem Sie ein paar Variablen ausgewählt haben, die einen Einfluss auf die »Kosten« haben, müssen Sie noch festlegen, wie Sie sie zu einer einzelnen Zahl zusammenführen. Hier ist es zum Beispiel notwendig zu entscheiden, wie viel Geld die Zeit im Flugzeug oder beim Warten auf dem Flughafen wert ist. Sie können vielleicht festlegen, dass es sich lohnt, 1 Dollar für jede Minute gesparter Flugzeit auszugeben (was bedeutet, zusätzliche 90 Dollar für einen Direktflug auszugeben, der anderthalb Stunden spart) und 0,50 Dollar für jede Minute, die man am Flughafen an Wartezeit einsparen kann. Sie können auch die Kosten für einen zusätzlichen Tag Automiete aufschlagen, wenn alle zu einem späteren Zeitpunkt zum Flughafen zurückfahren als, wie oben beschrieben, erforderlich.

Es gibt viele Möglichkeiten, wie die Funktion `getcost` hier definiert werden kann. Diese Funktion berücksichtigt die Gesamtflugzeit und die Gesamtzeit, die mit dem Warten im Flughafen auf die verschiedenen Familienmitglieder verbracht wird. Zudem fügt sie 50 Dollar Strafe hinzu, wenn das Auto zeitlich später am Tag

e-bol.net

zurückgegeben wird, als es am Ankunftstag abgeholt wurde. Fügen Sie diese Funktion der Datei *optimization.py* hinzu und schlagen Sie nach Wunsch zusätzliche Kosten auf bzw. passen Sie das Verhältnis zwischen Geld und Zeit an:

```
def schedulecost(sol):
  totalprice=0
  latestarrival=0
  earliestdep=24*60

  for d in range(len(sol)/2):
    # Hin- und Rückflüge holen.
    origin=people[d][1]
    outbound=flights[(origin,destination)][int(sol[d])]
    returnf=flights[(destination,origin)][int(sol[d+1])]

    # Der Gesamtpreis ist der Preis aller Hin- und Rückflüge.
    totalprice+=outbound[2]
    totalprice+=returnf[2]

    # Die letzte Ankunft und den ersten Abflug protokollieren.
    if latestarrival<getminutes(outbound[1]): latestarrival=getminutes(outbound[1])
    if earliestdep>getminutes(returnf[0]): earliestdep=getminutes(returnf[0])

  # Jede Person muss am Flughafen warten, bis die letzte Person angekommen ist.
  # Und sie müssen zur gleichen Zeit zurück sein und auf ihre Flüge warten.
  totalwait=0
  for d in range(len(sol)/2):
    origin=people[d][1]
    outbound=flights[(origin,destination)][int(sol[d])]
    returnf=flights[(destination,origin)][int(sol[d+1])]
    totalwait+=latestarrival-getminutes(outbound[1])
    totalwait+=getminutes(returnf[0])-earliestdep

  # Muss das Auto einen Tag länger gemietet werden? Das sind 50 Dollar mehr!
  if latestarrival>earliestdep: totalprice+=50

  return totalprice+totalwait
```

Die Logik dieser Funktion ist sehr simpel, aber sie zeigt, worum es geht. Sie kann an vielen Stellen verbessert werden – so wird bei der Gesamtwartezeit davon ausgegangen, dass alle Familienmitglieder den Flughafen gemeinsam erst dann verlassen, wenn die letzte Person angekommen ist, und wieder anreisen, bevor das erste Mitglied abfliegt. Das kann man so anpassen, dass jeder, der mehr als zwei Stunden warten würde, stattdessen sein eigenes Auto mietet, wodurch sich Kosten und Wartezeiten entsprechend anpassen würden.

Sie können diese Funktion in Ihrer Python-Session austesten:

```
>>> reload(optimization)
>>> optimization.schedulecost(s)
5285
```

e-bol.net

Nachdem die Zielfunktion nun erstellt wurde, sollte klar sein, dass es das Ziel ist, die Kosten zu minimieren, indem man die richtigen Werte für die Variablen wählt. Theoretisch könnten Sie versuchen, jede mögliche Kombination auszutesten, aber in diesem Beispiel gibt es 16 Flüge, die jeweils neun Möglichkeiten haben, wodurch man auf eine Gesamtsumme von 9^{16} (mehr als 1,8 Billiarden) Kombinationen kommt. Würden Sie jede Kombination ausprobieren, wäre sichergestellt, dass Sie garantiert die beste Antwort erhalten, aber es würde auf den meisten Rechnern ein sehr lange Zeit brauchen.

Random Search

Random Search ist keine sehr gute Optimierungsmethode, aber sie ermöglicht es, genau zu verstehen, was all die Algorithmen zu tun versuchen, und sie dient als Ausgangsbasis, sodass Sie vergleichen können, ob die anderen Algorithmen ihre Aufgabe gut erledigen.

Die Funktion erwartet ein paar Parameter. `Domain` ist eine Liste mit 2-Tupeln, die die minimalen und maximalen Werte für jede Variable festlegen. Die Länge der Lösung entspricht der Länge dieser Liste. Im aktuellen Beispiel gibt es neun Hin- und neun Rückflüge für jede Person, daher ist in `domain` für jede Person zweimal (0,8) aufgeführt.

Der zweite Parameter, `costf`, ist die Zielfunktion, in diesem Beispiel also `schedulecost`. Sie wird als Parameter übergeben, sodass die Funktion auch für andere Optimierungsprobleme genutzt werden kann. Diese Funktion erstellt zufällig 1.000 mögliche Lösungen und ruft für diese `costf` auf. Dann sucht sie den besten Vorschlag heraus (den mit den niedrigsten Kosten) und liefert ihn zurück. Fügen Sie die Funktion der Datei *optimization.py* hinzu:

```
def randomoptimize(domain,costf):
  best=999999999
  bestr=None
  for i in range(1000):
    # Eine zufällige Lösung erstellen.
    r=[random.randint(domain[i][0],domain[i][1])
       for i in range(len(domain))]

    # Kosten ermitteln
    cost=costf(r)

    # Mit dem bisher besten Ergebnis vergleichen.
    if cost<best:
      best=cost
      bestr=r
  return r
```

Natürlich sind 1.000 Versuche nur ein sehr kleiner Teil der möglichen Lösungen. Aber in diesem Beispiel gibt es viele Lösungen, das gut sind (wenn auch nicht die

e-bol.net

besten), daher ist die Wahrscheinlichkeit recht groß, mit dieser Funktion zu einem Ergebnis zu kommen, die nicht allzu schlimm ist. Testen Sie die Funktion in Ihrer Python-Sitzung aus:

```
>>> reload(optimization)
>>> domain=[(0,8)]*(len(optimization.people)*2)
>>> s=optimization.randomoptimize(domain,optimization.schedulecost)
>>> optimization.schedulecost(s)
3328
>>> optimization.printschedule(s)
   Seymour    Boston 12:34-15:02 $109 12:08-14:05 $142
    Franny    Dallas 12:19-15:25 $342  9:49-13:51 $229
     Zooey     Akron  9:15-12:14 $247 15:50-18:45 $243
      Walt     Miami 15:34-18:11 $326 14:08-16:09 $232
     Buddy   Chicago 14:22-16:32 $126 15:04-17:23 $189
       Les     Omaha 15:03-16:42 $135  6:19- 8:13 $239
```

Aufgrund der Zufallskomponente werden sich Ihre Ergebnisse von den hier gezeigten unterscheiden. Die Ergebnisse sind nicht super, da Zooey am Flughafen sechs Stunden warten muss, bis Walt ankommt, aber es könnte auf jeden Fall schlimmer sein. Lassen Sie die Funktion mehrere Male laufen, um zu sehen, ob sich die Kosten stark ändern, oder erhöhen Sie die Schleife auf 10.000 Durchläufe, um zu prüfen, ob Sie damit bessere Ergebnisse erhalten.

Bergsteigen

Zufällig verschiedene Lösungen auszutesten ist sehr ineffizient, weil dieses Verfahren keinen Vorteil aus der guten Lösung zieht, die schon gefunden wurde. In unserem Beispiel ähnelt ein Flugplan mit geringen Gesamtkosten sehr wahrscheinlich einem anderen Flugplan, der auch niedrige Kosten hat. Da bei der Zufallsoptimierung herumgesprungen wird, werden auch nicht automatisch ähnliche Flugpläne gesucht, die nahe bei den guten liegen, die schon gefunden wurden.

Eine alternative Methode zum Random Search wird als *Bergsteigeralgorithmus* (*Hill Climbing*) bezeichnet. Dabei wird mit einer zufälligen Lösung begonnen; anschließend werden die Nachbarlösungen nach einer durchsucht, die besser ist (also eine niedrigere Zielfunktion hat). Das ist analog zu Heruntersteigen von einem Hügel, wie in Abbildung 5-1 dargestellt.

Versetzen Sie sich einmal in die Person auf dem Bild: Sie werden zufällig in der Landschaft abgesetzt. Sie wollen den tiefsten Punkt finden, um nach Wasser zu suchen. Dazu schauen Sie vielleicht in jede Richtung und gehen dorthin, wo das Gelände am steilsten nach unten geht. Sie setzen dann Ihren Weg so lange fort, bis Sie einen Punkt erreicht haben, an dem das Gelände flach bleibt oder wieder ansteigt.

e-bol.net

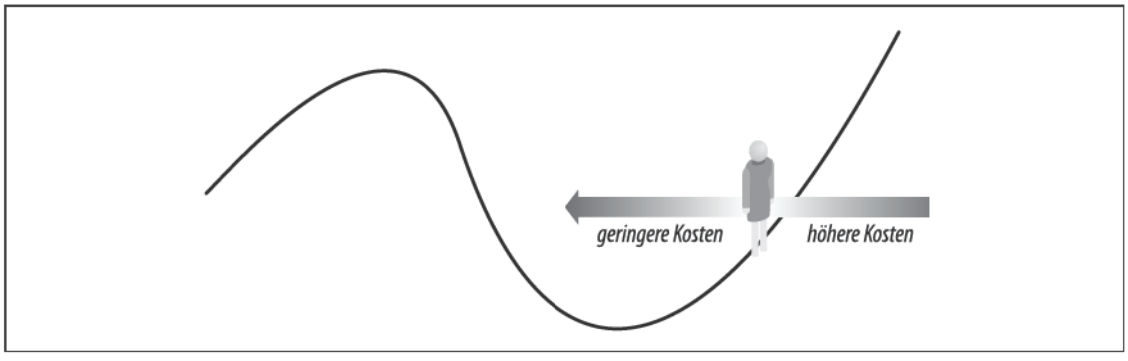

Abbildung 5-1: Die niedrigsten Kosten auf einem Hügel suchen

Sie können den Bergsteiger-Ansatz auf die Suche nach dem besten Flugplan für die Familie Glass anwenden. Beginnen Sie mit einem zufälligen Flugplan und suchen Sie dann alle benachbarten Flugpläne heraus. In diesem Fall bedeutet das, alle Flugpläne zu finden, bei der eine Person einen etwas früheren oder späteren Flug nehmen würde. Die Kosten werden für jeden der Nachbar-Flugpläne berechnet und derjenige mit den niedrigsten Kosten wird der neue Flugplan. Dieser Prozess wird wiederholt, bis keiner der Nachbar-Flugpläne die Kosten weiter reduziert.

Zum Implementieren fügen Sie `hillclimb` der Datei *optimization.py* hinzu:

```
def hillclimb(domain,costf):
  # Eine zufällige Lösung erstellen.
  sol=[random.randint(domain[i][0],domain[i][1])
     for i in range(len(domain))]

  # Hauptschleife
  while 1:

    # Liste mit benachbarten Lösungen finden.
    neighbors=[]
    for j in range(len(domain)):

      # Einen weiter in jede Richtung.
      if sol[j]>domain[j][0]:
        neighbors.append(sol[0:j]+[sol[j]+1]+sol[j+1:])
      if sol[j]<domain[j][1]:
        neighbors.append(sol[0:j]+[sol[j]-1]+sol[j+1:])

    # Die beste Lösung unter den Nachbarn finden.
    current=costf(sol)
    best=current
    for j in range(len(neighbors)):
      cost=costf(neighbors[j])
      if cost<best:
        best=cost
        sol=neighbors[j]

    # Wenn es keine Verbesserung gibt, sind wir am Ziel!
```

e-bol.net

```
        if best==current:
          break

    return sol
```

Diese Funktion erzeugt eine zufällige Liste mit Zahlen innerhalb der gegebenen Domäne, um die erste Lösung zu erstellen. Dann findet sie alle Nachbarn für die aktuelle Lösung, indem sie jedes Element in der Liste durchläuft und zwei neue Listeneinträge mit diesem Element um eins erhöht beziehungsweise reduziert anlegt. Die beste dieser Nachbar-Lösungen wird die neue Lösung.

Testen Sie diese Funktion in Ihrer Python-Session aus, um zu sehen, wie sie sich gegenüber der Random Search-Methode schlägt:

```
>>> s=optimization.hillclimb(domain,optimization.schedulecost)
>>> optimization.schedulecost(s)
3063
>>> optimization.printschedule(s)
   Seymour       BOS 12:34-15:02 $109 10:33-12:03 $ 74
    Franny       DAL 10:30-14:57 $290 10:51-14:16 $256
     Zooey       CAK 10:53-13:36 $189 10:32-13:16 $139
      Walt       MIA 11:28-14:40 $248 12:37-15:05 $170
     Buddy       ORD 12:44-14:17 $134 10:33-13:11 $132
       Les       OMA 11:08-13:07 $175 18:25-20:34 $205
```

Diese Funktion läuft schnell durch und findet normalerweise eine bessere Lösung als die Random Search. Es gibt allerdings einen großen Nachteil beim Bergsteigeralgorithmus. Schauen Sie sich Abbildung 5-2 an.

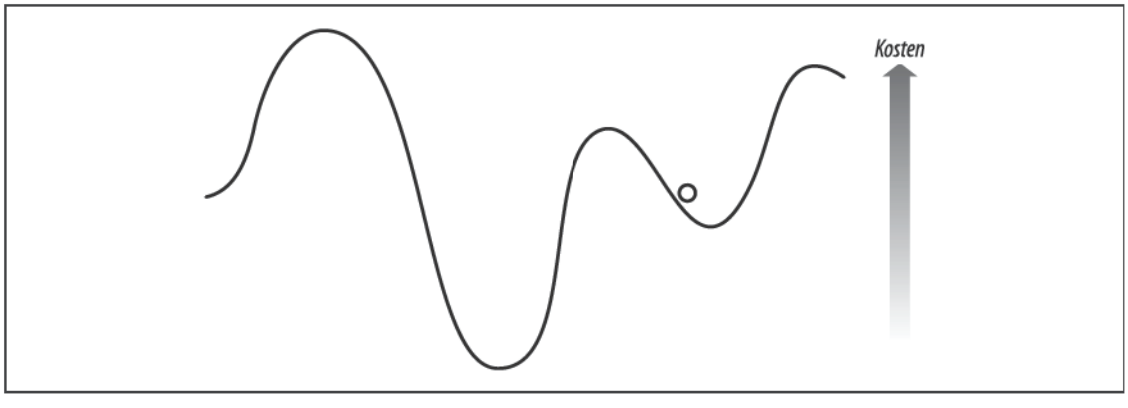

Abbildung 5-2: Gefangen in einem lokalen Minimum

In dieser Abbildung sieht man deutlich, dass eine einfache Bewegung den Hang hinunter nicht notwendigerweise zur insgesamt besten Lösung führen wird. Die schließlich gefundene Lösung wird ein *lokales Minimum* sein – eine Lösung, die besser als die in der Umgebung sind, aber nicht die insgesamt beste. Die insgesamt beste Lösung wird als *globales Minimum* bezeichnet, dessen Finden das eigentliche Ziel von Optimierungsalgorithmen ist. Ein Ansatz, dieses Dilemma zu lösen, wird als *Bergsteigeralgorithmus mit zufälligem neuem Startpunkt* (*Random-Restart Hill*

e-bol.net

Climbing) bezeichnet. Dabei wird der Bergsteigeralgorithmus viele Male mit zufälligen Ausgangspunkten durchlaufen – in der Hoffnung, dass einer der Durchläufe nahe am globalen Minimum liegen wird. Die nächsten zwei Abschnitte »Simulierte Abkühlung« und »Genetische Algorithmen« stellen andere Möglichkeiten vor, um zu vermeiden, in einem lokalen Minimum stecken zu bleiben.

Simulierte Abkühlung

Die *simulierte Abkühlung* (*Simulated Annealing*) ist eine Optimierungsmethode, die physikalischen Prozessen nachempfunden ist. Beim Abkühlen (oder eher Ausglühen) wird eine Legierung erhitzt, die dann langsam erkaltet. Da die Atome zuerst wild herumhüpfen und sich dann nach und nach in einem Zustand mit niedriger Energie beruhigen, können sie eine Konfiguration mit niedriger Energie finden.

Die Algorithmusversion der Abkühlung startet mit einer zufälligen Lösung des Problems. Sie nutzt eine Variable, die die Temperatur repräsentiert und mit einem hohen Wert beginnt, der nach und nach niedriger wird. In jedem Durchlauf wird eine der Zahlen aus der Lösung zufällig ausgewählt und in eine bestimmte Richtung geändert. In unserem Beispiel könnte der Rückflug von Seymour vom zweiten Flug des Tages auf den dritten verschoben werden. Die Kosten werden vor und nach der Änderung berechnet und dann miteinander verglichen.

Hier kommt der wichtige Teil: Wenn die neuen Kosten geringer sind, wird die neue Lösung zur aktuellen Lösung, also wie bei der Bergsteigermethode. Wenn die Kosten allerdings *höher* sind, kann die neue Lösung immer noch mit einer gewissen Wahrscheinlichkeit zur aktuellen Lösung werden. Dies ist ein Versuch, das Problem mit dem lokalen Minimum zu vermeiden, das in Abbildung 5-2 aufgezeigt wurde.

In manchen Fällen ist es notwendig, sich in Richtung einer schlechteren Lösung zu bewegen, bevor Sie von dort eine noch bessere erreichen können. Die simulierte Abkühlung funktioniert, weil sie immer eine Bewegung zum Besseren hin akzeptiert und zu Beginn des Algorithmus auch eine schlechtere Lösung erträgt. Wenn der Prozess fortschreitet, wird es mehr und mehr unwahrscheinlich, dass eine schlechtere Lösung akzeptiert wird, bis am Ende nur noch bessere Lösungen in Frage kommen. Die Wahrscheinlichkeit einer teureren Lösung, die akzeptiert wird, ist durch diese Formel angegeben:

$$p=e^{((-highcost-lowcost)/temperature)}$$

Da die Temperatur (die Bereitschaft, eine schlechtere Lösung zu akzeptieren) mit einem sehr hohen Wert beginnt, wird der Exponent immer nahe bei 0 liegen, womit die Wahrscheinlichkeit fast 1 sein wird. Sobald die Temperatur fällt, wird der Unterschied zwischen der teureren und der billigeren Lösung wichtiger – ein größerer Unterschied führt zu einer kleineren Wahrscheinlichkeit, daher wird der Algorithmus nur wenig schlechtere Lösungen akzeptieren.

e-bol.net

Erstellen Sie eine neue Funktion in *optimization.py* mit dem Namen `annealingoptimize`, die den Algorithmus implementiert:

```
def annealingoptimize(domain,costf,T=10000.0,cool=0.95,step=1):
  # Die Werte zufällig initialisieren.
  vec=[float(random.randint(domain[i][0],domain[i][1]))
       for i in range(len(domain))]

  while T>0.1:
    # Einen der Indexe wählen.
    i=random.randint(0,len(domain)-1)

    # Eine Änderungsrichtung wählen.
    dir=random.randint(-step,step)

    # Eine neue Liste erstellen, bei der einer der Werte geändert wurde.
    vecb=vec[:]
    vecb[i]+=dir
    if vecb[i]<domain[i][0]: vecb[i]=domain[i][0]
    elif vecb[i]>domain[i][1]: vecb[i]=domain[i][1]

    # Berechnen der aktuellen und der neuen Kosten.
    ea=costf(vec)
    eb=costf(vecb)
    p=pow(math.e,(-eb-ea)/T)

    # Ist es besser, oder hilft die Wahrscheinlichkeit nach?
    if (eb<ea or random.random( )<p):
      vec=vecb

    # Temperatur reduzieren.
    T=T*cool
  return vec
```

Diese Funktion erstellt zunächst eine zufällige Lösung mit der richtigen Länge, wobei alle Werte im durch den Parameter `domain` definierten Bereich liegen. Die `temperature` und die Abkühlungsrate `cooling` sind optionale Parameter. In jeder Iteration wird `i` auf einen zufälligen Index der Lösung gesetzt und `dir` auf eine Zufallszahl zwischen –`step` und `step`. Dann berechnet die Funktion die aktuellen Kosten und die Kosten, die entstehen würden, wenn man den Wert bei `i` um `dir` änderte.

Die fett gedruckte Codezeile zeigt die Berechnung der Wahrscheinlichkeit, die sinkt, sobald `T` sinkt. Wenn eine Zufallsgleitkommazahl zwischen 0 und 1 kleiner als dieser Wert ist oder wenn die neue Lösung besser ist, akzeptiert die Funktion die neue Lösung. Diese Iterationsschritte werden durchlaufen, bis die Temperatur fast 0 ist, wobei sie jedes Mal mit der Abkühlungsrate multipliziert wird.

Jetzt können Sie in Ihrer Python-Session mit der simulierten Abkühlung eine Optimierung versuchen:

```
>>> reload(optimization)
>>> s=optimization.annealingoptimize(domain,optimization.schedulecost)
```

e-bol.net

```
>>> optimization.schedulecost(s)
2278
>>> optimization.printschedule(s)
   Seymour    Boston 12:34-15:02 $109 10:33-12:03 $ 74
    Franny    Dallas 10:30-14:57 $290 10:51-14:16 $256
     Zooey     Akron 10:53-13:36 $189 10:32-13:16 $139
      Walt     Miami 11:28-14:40 $248 12:37-15:05 $170
     Buddy   Chicago 12:44-14:17 $134 10:33-13:11 $132
       Les     Omaha 11:08-13:07 $175 15:07-17:21 $129
```

Diese Optimierung hat eine gute Lösung gefunden: Die Gesamtwartezeiten sind gering, die Kosten bleiben im Rahmen. Natürlich werden sich Ihre Ergebnisse von diesen unterscheiden, und es kann sein, dass sie schlechter sind. Es ist bei jedem Problem gut, mit verschiedenen Parametern für die Ausgangstemperatur und die Abkühlungsrate zu experimentieren. Sie können auch die Schrittweite für die Zufallsbewegungen variieren.

Genetische Algorithmen

Eine andere Gruppe von Techniken für die Optimierung, ebenfalls der Natur abgeschaut, wird als *genetische Algorithmen* bezeichnet. Dabei wird zunächst eine Reihe von zufälligen Lösungen erstellt, die *Population* genannt werden. Bei jedem Optimierungsschritt wird die Zielfunktion für die gesamte Population berechnet, um eine Bewertungsreihenfolge für die Lösungen zu erhalten. Ein Beispiel ist in Tabelle 5-1 zu sehen.

Tabelle 5-1: Bewertete Liste mit Lösungen und Kosten

Lösung	Kosten
[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3]	4394
[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8]	4661
…	…
[0, 4, 0, 3, 8, 8, 4, 4, 8, 5, 6, 1]	7845
[5, 8, 0, 2, 8, 8, 8, 2, 1, 6, 6, 8]	8088

Nach der Bewertung der Lösungen wird eine neue Population – bezeichnet als nächste *Generation* – erstellt. Dabei werden die besten Lösungen der aktuellen Population der neuen Population unverändert hinzugefügt. Dieser Prozess wird als *Selektion* bezeichnet. Der Rest der neuen Population besteht aus vollständig neuen Lösungen, die durch Veränderung der besten Lösungen gewonnen werden.

Es gibt zwei Wege, die Lösungen zu verändern. Der einfachere wird als *Mutation* bezeichnet, was normalerweise eine kleine, einfache und zufällige Änderung der bestehenden Lösung bedeutet. In diesem Fall kann eine Mutation einfach vorge-

e-bol.net

nommen werden, indem man sich eine der Zahlen aus der Lösung nimmt und sie erhöht oder verringert. Ein paar Beispiele werden in Abbildung 5-3 gezeigt.

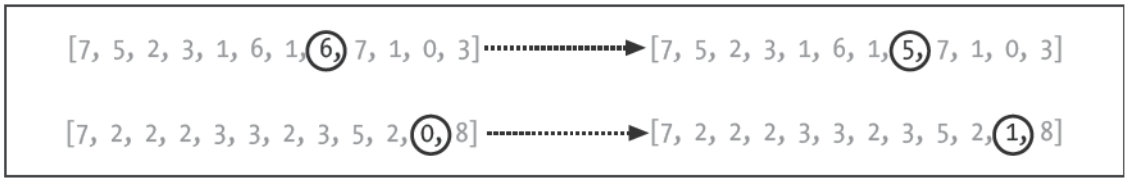

Abbildung 5-3: Beispiele für die Mutation einer Lösung

Die andere Möglichkeit, Lösungen zu verändern, wird als *Crossover* oder *Breeding* (*Zucht*) bezeichnet. Dabei werden zwei der besten Lösungen genommen und auf eine bestimmte Art und Weise kombiniert. In diesem Fall wäre eine einfache Form des Crossover die Wahl einer zufälligen Zahl von Elementen aus einer Lösung, um sie mit den restlichen Elementen einer anderen Lösung zu kombinieren, wie dies in Abbildung 5-4 zu sehen ist.

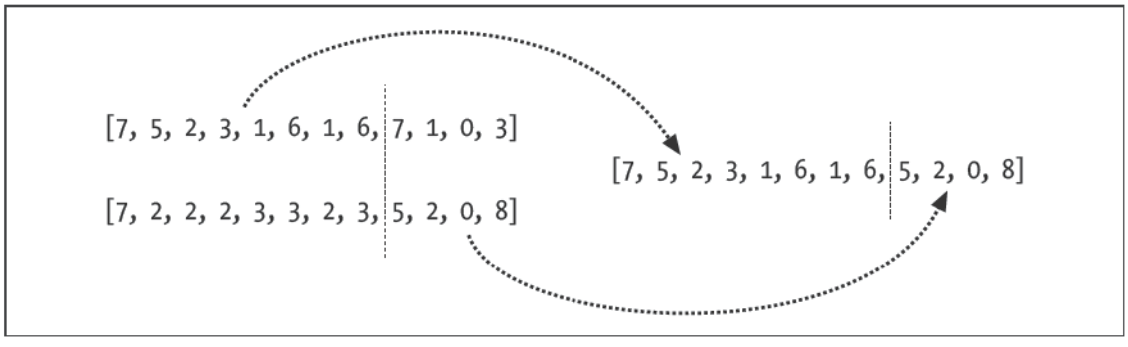

Abbildung 5-4: Beispiel für ein Crossover

Eine neue Population – normalerweise in gleicher Größe wie die alte – wird erstellt, indem die besten Lösungen zufällig mutiert und »gezüchtet« werden. Dann wiederholt sich der Prozess – die neue Population wird bewertet und erneut eine neue Population erzeugt. Das setzt sich fort, bis entweder ein feste Zahl von Iterationen durchlaufen ist oder es über mehrere Generationen keine Verbesserung mehr gibt.

Fügen Sie `geneticoptimize` der Datei *optimization.py* hinzu:

```
def geneticoptimize(domain,costf,popsize=50,step=1,
                    mutprod=0.2,elite=0.2,maxiter=100):
  # Mutation
  def mutate(vec):
    i=random.randint(0,len(domain)-1)
    if random.random( )<0.5 and vec[i]>domain[i][0]:
      return vec[0:i]+[vec[i]-step]+vec[i+1:]
    elif vec[i]<domain[i][1]:
      return vec[0:i]+[vec[i]+step]+vec[i+1:]

  # Crossover
  def crossover(r1,r2):
```

e-bol.net

```
    i=random.randint(1,len(domain)-2)
    return r1[0:i]+r2[i:]

  # Erste Population aufbauen.
  pop=[]
  for i in range(popsize):
    vec=[random.randint(domain[i][0],domain[i][1])
         for i in range(len(domain))]
    pop.append(vec)

  # Wie viele Gewinner in jeder Generation?
  topelite=int(elite*popsize)

  # Hauptschleife
  for i in range(maxiter):
    scores=[(costf(v),v) for v in pop]
    scores.sort( )
    ranked=[v for (s,v) in scores]

    # Nur mit den Gewinnern beginnen.
    pop=ranked[0:topelite]

    # Mutierte und gezüchtete Formen der Gewinner hinzufügen.
    while len(pop)<popsize:
      if random.random( )<mutprob:

        # Mutation
        c=random.randint(0,topelite)
        pop.append(mutate(ranked[c]))
      else:

        # Crossover
        c1=random.randint(0,topelite)
        c2=random.randint(0,topelite)
        pop.append(crossover(ranked[c1],ranked[c2]))

    # Aktuell besten Wert ausgeben.
    print scores[0][0]

  return scores[0][1]
```

Diese Funktion hat diverse optionale Parameter:

`popsize`
: Die Größe der Population.

`mutprob`
: Die Wahrscheinlichkeit, dass ein neues Element der Population eine Mutation und kein Crossover sein wird.

`elite`
: Der Anteil der Population, der als gute Lösung angesehen wird und der es in die nächste Generation schafft.

e-bol.net

`maxiter`
: Die Anzahl an Generationen.

Versuchen Sie, die Flugpläne mithilfe des genetischen Algorithmus in Ihrer Python-Session zu optimieren:

```
>>> s=optimization.geneticoptimize(domain,optimization.schedulecost)
3532
3503
...
2591
2591
2591
>>> optimization.printschedule(s)
   Seymour       BOS 12:34-15:02 $109 10:33-12:03 $ 74
    Franny       DAL 10:30-14:57 $290 10:51-14:16 $256
     Zooey       CAK 10:53-13:36 $189 10:32-13:16 $139
      Walt       MIA 11:28-14:40 $248 12:37-15:05 $170
     Buddy       ORD 12:44-14:17 $134 10:33-13:11 $132
       Les       OMA 11:08-13:07 $175 11:07-13:24 $171
```

In Kapitel 11 werden Sie eine Erweiterung der genetischen Algorithmen namens *genetische Programmierung* kennenlernen, bei der mit ähnlichen Ideen komplett neue Programme erstellt werden.

Der Computerwissenschaftler John Holland wird allgemein als Vater der genetischen Algorithmen angesehen, denn er ist der Autor des 1975 erschienenen Buchs *Adaptation in Natural and Artificial Systems* (University of Michigan Press). Seine Arbeiten gehen zurück auf Versuche von Biologen in den 1950er-Jahren, die Evolution auf dem Computer zu modellieren. Seitdem wurden genetische Algorithmen und andere Optimierungsmethoden für viele verschiedene Probleme genutzt, so zum Beispiel zum:

- Ermitteln, welche Form eines Konzertsaals die beste Akustik bietet,
- Entwerfen eines optimalen Flügels für ein Überschallflugzeug,
- Vorschlagen der besten Sammlung an Chemikalien, um sie als potenzielle Drogen zu untersuchen
- sowie zum automatischen Designen eines Chips zur Spracherkennung.

Mögliche Lösungen für diese Probleme lassen sich in Listen von Zahlen umwandeln. Damit wird es einfach, genetische Algorithmen oder die simulierte Abkühlung anzuwenden.

Ob eine bestimmte Optimierungsmethode funktioniert, hängt sehr vom Problem ab. Die simulierte Abkühlung, genetische Optimierungen und die meisten anderen Optimierungsmethoden hängen stark von der Tatsache ab, dass die beste Lösung in den meisten Fällen nahe an anderen guten Lösungen liegt. Ein Fall, in dem die Optimierung eher nicht gut funktionieren wird, ist in Abbildung 5-5 abgebildet.

e-bol.net

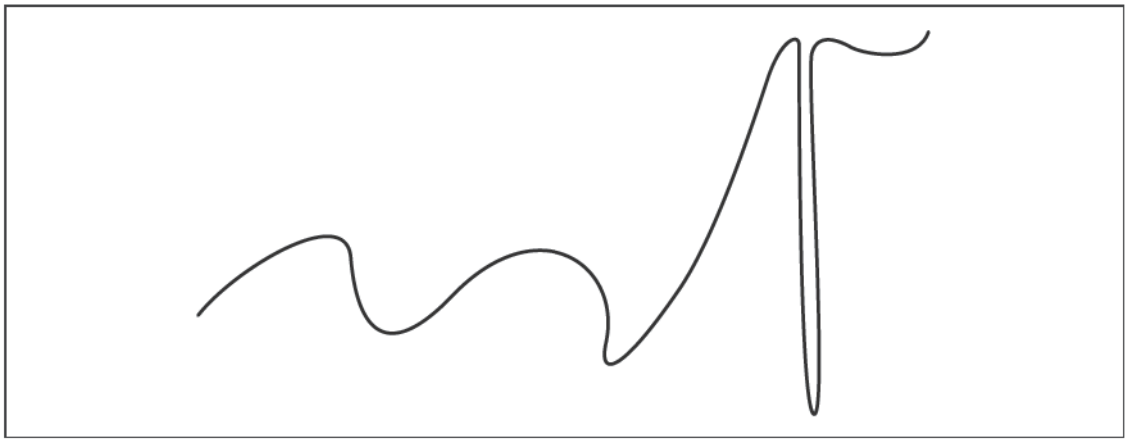

Abbildung 5-5: Schlechtes Problem für die Optimierung

Die Kosten sind an einer sehr steilen Stelle ziemlich weit rechts in der Abbildung am niedrigsten. Jede Lösung, die dort in der Nähe liegt, wird vermutlich verworfen werden, weil die Kosten zu hoch sind, und Sie werden dieses globale Minimum niemals finden. Die meisten Algorithmen würden sich auf eines der lokalen Minima auf der linken Seite der Abbildung einschießen.

Das Beispiel mit dem Flugplan funktioniert, weil das Versetzen einer Person vom zweiten auf den dritten Flug des Tages die Gesamtkosten vermutlich weniger ändern wird als das Versetzen der Person auf den achten Flug des Tages. Wenn die Flüge zufällig in der Liste eingeordnet wären, würden die Optimierungsmethoden nicht besser als eine Random Search funktionieren – und in der Tat gibt es keine Optimierungsmethode, die in diesem Fall durchgängig besser funktioniert als Random Search.

Echte Flugsuchen

Nachdem nun mit den Beispieldaten alles läuft, ist es an der Zeit zu versuchen, echte Flugdaten zu holen, um zu sehen, ob dort die gleichen Optimierungen durchgeführt werden können. Sie werden Daten von Kayak herunterladen, das eine API für Flugsuchen bereitstellt. Der Hauptunterschied zwischen echten Flugdaten und den Beispieldaten, mit denen Sie gearbeitet haben, ist, dass es bei den realen Flugdaten deutlich mehr als neun Flüge pro Tag zwischen den meisten wichtigen Städten gibt.

Die Kayak-API

Kayak, von dem Sie eine Webseite in Abbildung 5-6 sehen, ist eine beliebte *vertikale Such-Engine* für Reisen. Auch wenn es eine ganze Reihe von Reisesites online gibt, ist Kayak für dieses Beispiel nützlich, da es eine nette XML-API hat, die genutzt werden kann, um echte Reisesuchen aus einem Python-Programm heraus durchzu-

e-bol.net

führen. Um die API zu nutzen, müssen Sie sich für einen Entwicklerschlüssel auf der Seite *http://www.kayak.com/labs/api/search* anmelden.

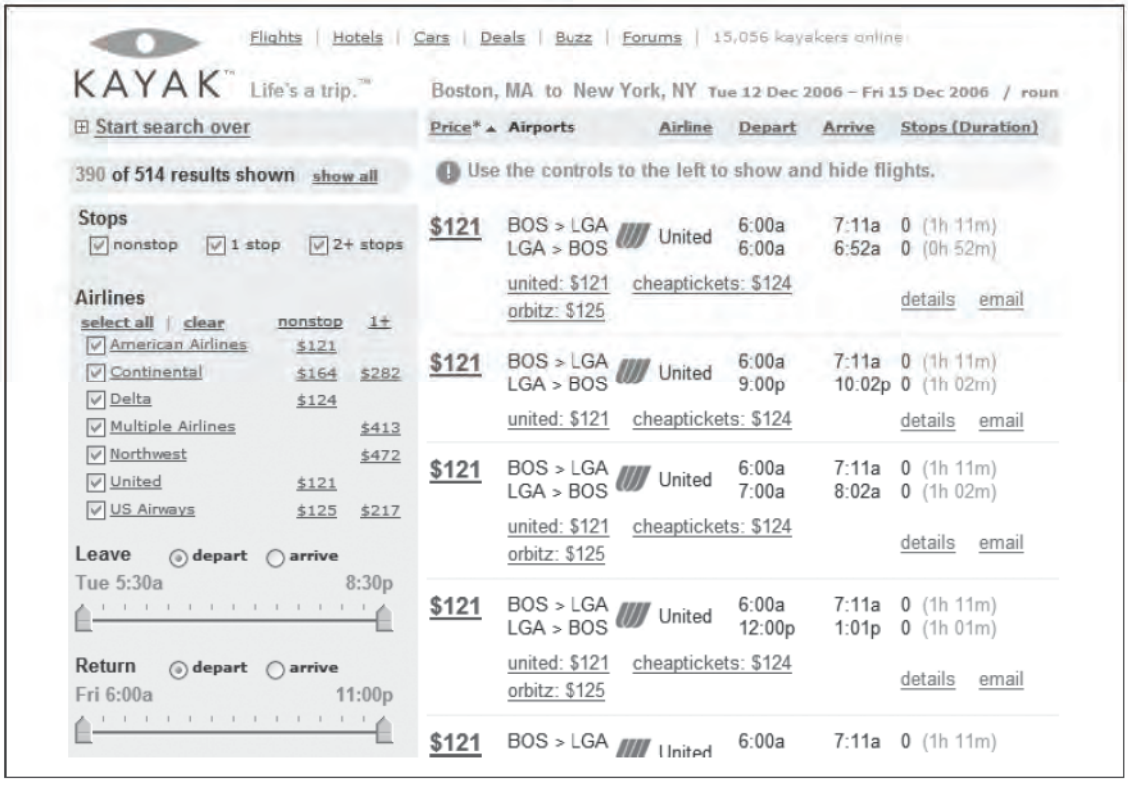

Abbildung 5-6: Screenshot der Reise-Suchschnittstelle von Kayak

Der Entwicklerschlüssel ist ein langer String aus Zahlen und Buchstaben, den Sie nutzen müssen, um in Kayak Flugsuchen durchzuführen (man kann ihn auch zur Hotelsuche benutzen, aber darum wollen wir uns hier nicht kümmern). Zum Zeitpunkt der Entstehung dieses Buchs gibt es keine spezifische Python-API für Kayak, so wie es sie für *del.icio.us* gibt, aber die XML-Schnittstelle ist sehr gut erläutert. Dieses Kapitel wird Ihnen zeigen, wie Sie mithilfe der Python-Pakete *urllib2* und *xml.dom.minidom* suchen können, wobei beide Pakete zur Standard-Python-Distribution gehören.

Das Paket minidom

Das Paket *minidom* ist Teil der Standard-Python-Distribution. Es handelt sich um eine schlanke Implementierung der DOM-(Document Object Model-)Schnittstelle, einer Standardvorgehensweise, um ein XML-Dokument als Baum mit Objekten zu behandeln. Das Paket erwartet Strings oder geöffnete Dateien, die XML enthalten, und liefert ein Objekt zurück, mit dem Sie die Informationen einfach extrahieren können. Geben Sie zum Beispiel das Folgende in einer Python-Session ein:

e-bol.net

```
>>> import xml.dom.minidom
>>> dom=xml.dom.minidom.parseString('<data><rec>Hallo!</rec></data>')
>>> dom
<xml.dom.minidom.Document instance at 0x00980C38>
>>> r=dom.getElementsByTagName('rec')
>>> r
[<DOM Element: rec at 0xa42350>]
>>> r[0].firstChild
<DOM Text node "Hallo!">
>>> r[0].firstChild.data
u'Hallo!'
```

Da viele Websites mittlerweile eine Möglichkeit anbieten, auf Informationen über eine XML-Schnittstelle zuzugreifen, ist es sehr nützlich zu lernen, wie die Python-XML-Pakete zu gebrauchen sind, wenn man kollektive Intelligenz programmieren will. Hier die wichtigsten Methoden von DOM-Objekten, die Sie mit der Kayak-API nutzen werden:

`getElementsByTagName(name)`
: Liefert eine Liste aller DOM-Knoten zurück, indem das gesamte Dokument nach Elementen durchforstet wird, deren Tags zu `name` passen.

`firstChild`
: Gibt das erste Kindelement dieses Objekts zurück. Im obigen Beispiel ist das erste Kindelement von `rec` der Knoten, der den Text »Hallo!« repräsentiert.

`data`
: Liefert die Daten zurück, die mit diesem Objekt verbunden sind – in den meisten Fällen ein Unicode-String mit dem Text, der im Knoten enthalten ist.

Flugsuchen

Beginnen Sie mit dem Anlegen einer neuen Datei namens *kayak.py* und fügen Sie die folgenden Anweisungen hinzu:

```
import time
import urllib2
import xml.dom.minidom

kayakkey='YOURKEYHERE'
```

Als Erstes müssen Sie nun Code schreiben, um eine neue Kayak-Session mit Ihrem Entwicklerschlüssel zu erhalten. Die dafür gedachte Funktion schickt einen Request an `apisession`, wobei der Parameter `token` auf Ihren Entwicklerschlüssel gesetzt ist. Das von dieser URL zurückgelieferte XML wird ein Tag `sid` mit einer Session-ID enthalten:

```
<sid>1-hX4lII_wS$8b06a07kHj</sid>
```

Die Funktion muss nur das XML parsen, um den Inhalt des Tags `sid` auszulesen. Fügen Sie diese Funktion *kayak.py* hinzu:

e-bol.net

```
def getkayaksession( ):
  # URL zum Start einer Session zusammensetzen.
  url='http://www.kayak.com/k/ident/apisession?token=%s&version=1' % kayakkey

  # Das zurückgegebene XML parsen.
  doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))

  # <sid>xxxxxxxx</sid> finden.
  sid=doc.getElementsByTagName('sid')[0].firstChild.data
  return sid
```

Als Nächstes erstellen Sie eine Funktion, um die Flugsuche auszuführen. Die URL für diese Suche ist sehr lang, da sie alle Parameter für die Flugsuche enthält. Die wichtigsten Parameter für die Suche sind `sid` (die von `getkayaksession` zurückgegebene Session-ID), `destination` und `depart_date`.

Das zurückgeschickte XML-Dokument hat ein Tag namens `searchid`, das die Funktion genau so extrahiert, wie dies bei `getkayaksession` geschieht. Da die Suche eine lange Zeit in Anspruch nehmen kann, liefert dieser Aufruf keine echten Ergebnisse zurück – er beginnt nur mit der Suche und gibt eine ID an, mit der die Ergebnisse abgefragt werden können.

Fügen Sie diese Funktion der Datei *kayak.py* hinzu:

```
def flightsearch(sid,origin,destination,depart_date):

  # Such-URL
  url='http://www.kayak.com/s/apisearch?basicmode=true&oneway=y&origin=%s' % origin
  url+='&destination=%s&depart_date=%s' % (destination,depart_date)
  url+='&return_date=none&depart_time=a&return_time=a'
  url+='&travelers=1&cabin=e&action=doFlights&apimode=1'
  url+='&_sid_=%s&version=1' % (sid)

  # XML holen.
  doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))

  # Such-ID ermitteln.
  searchid=doc.getElementsByTagName('searchid')[0].firstChild.data

  return searchid
```

Schließlich brauchen Sie noch eine Funktion, die die Ergebnisse abfragt, bis es keine weiteren mehr gibt. Kayak stellt eine weitere URL `flight` zur Verfügung, die diese Daten liefert. Im zurückgegebenen XML gibt es das Tag `morepending`, das das Wort »true« enthält, bis die Suche abgeschlossen ist. Die Funktion muss die Seite abfragen, bis `morepending` nicht mehr true ist, dann hat die Funktion die vollständigen Ergebnisse.

Fügen Sie diese Funktion *kayak.py* hinzu:

```
def flightsearchresults(sid,searchid):

  # Führendes $ und Kommata entfernen, dann in ein float umwandeln.
```

e-bol.net

```
def parseprice(p):
  return float(p[1:].replace(',',''))

# Polling-Schleife
while 1:
  time.sleep(2)

  # URL zum Pollen bauen.
  url='http://www.kayak.com/s/basic/flight?'
  url+='searchid=%s&c=5&apimode=1&_sid_=%s&version=1' % (searchid,sid)
  doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))

  # Nach Tag morepending suchen und warten, bis es nicht mehr true ist.
  morepending=doc.getElementsByTagName('morepending')[0].firstChild
  if morepending==None or morepending.data=='false': break

# Jetzt die komplette Liste herunterladen.
url='http://www.kayak.com/s/basic/flight?'
url+='searchid=%s&c=999&apimode=1&_sid_=%s&version=1' % (searchid,sid)
doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))

# Elemente als Liste zusammenstellen.
prices=doc.getElementsByTagName('price')
departures=doc.getElementsByTagName('depart')
arrivals=doc.getElementsByTagName('arrive')

# Zusammenpacken.
return zip([p.firstChild.data.split(' ')[1] for p in departures],
           [p.firstChild.data.split(' ')[1] for p in arrivals],
           [parseprice(p.firstChild.data) for p in prices])
```

Beachten Sie, dass die Funktion am Ende einfach alle Tags `price`, `depart` und `arrive` zusammensucht. Von jedem Tag gibt es drei gleich viele Einträge – eines für jeden Flug –, sodass die Funktion `zip` genutzt werden kann, um alle in Tupel in einer großen Liste zusammenzufassen. Die Abflugs- und Ankunftsinformationen werden als Datum und Uhrzeit (getrennt durch ein Leerzeichen) übertragen, daher teilt die Funktion den String auf, um nur die Uhrzeit zu erhalten. Sie wandelt zudem den Preis in eine Gleitkommazahl um, indem sie ihn an `parseprice` übergibt.

Sie können eine echte Flugsuche in Ihrer Python-Session durchführen, um sicherzustellen, dass alles funktioniert (denken Sie daran, das Datum auf eines zu ändern, das in der Zukunft liegt):

```
>>> import kayak
>>> sid=kayak.getkayaksession( )
>>> searchid=kayak.flightsearch(sid,'BOS','LGA','11/17/2006')
>>> f=kayak.flightsearchresults(sid,searchid)
>>> f[0:3]
[(u'07:00', u'08:25', 60.3),
 (u'08:30', u'09:49', 60.3),
 (u'06:35', u'07:54', 65.0)]
```

e-bol.net

Flüge werden praktischerweise nach Preisen und dann nach Uhrzeiten geordnet zurückgegeben. Das funktioniert gut, da wie zuvor ähnliche Lösungen nahe beieinanderliegen. Die einzige Anforderung, um dies mit dem Rest des Codes zusammenzubringen, ist, einen vollständigen Flugplan für alle Mitglieder der Familie Glass in der Struktur zu erstellen, in der die Daten auch aus der Datei geladen wurden. Dafür muss man nur über die Personen in der Liste iterieren und die Flugsuche für ihre Hin- und Rückflüge durchführen. Fügen Sie die Funktion `createschedule` der Datei *kayak.py* hinzu:

```
def createschedule(people,dest,dep,ret):
  # Session-ID für die Suchen holen.
  sid=getkayaksession()
  flights={}

  for p in people:
    name,origin=p
    # Hinflüge
    searchid=flightsearch(sid,origin,dest,dep)
    flights[(origin,dest)]=flightsearchresults(sid,searchid)

    # Rückflüge
    searchid=flightsearch(sid,dest,origin,ret)
    flights[(dest,origin)]=flightsearchresults(sid,searchid)

  return flights
```

Jetzt können Sie versuchen, die Flüge für die Familie mit echten Flugdaten zu optimieren. Die Kayak-Suchen können eine Weile dauern, daher sollten Sie die Suche zu Beginn auf die ersten beiden Familienmitglieder beschränken. Geben Sie dies in Ihrer Python-Session ein:

```
>>> reload(kayak)
>>> f=kayak.createschedule(optimization.people[0:2],'LGA',
... '11/17/2006','11/19/2006')
>>> optimization.flights=f
>>> domain=[(0,30)]*len(f)
>>> optimization.geneticoptimize(domain,optimization.schedulecost)
770.0
703.0
...
>>> optimization.printschedule(s)
   Seymour       BOS 16:00-17:20 $85.0 19:00-20:28 $65.0
    Franny       DAL 08:00-17:25 $205.0 18:55-00:15 $133.0
```

Glückwunsch! Sie haben gerade eine Optimierung mit echten Flugdaten vorgenommen. Der Suchraum ist viel größer, daher sollten Sie mit der maximalen Geschwindigkeit und Lernrate experimentieren.

Es gibt viele Möglichkeiten der Erweiterung. Sie könnten zum Beispiel eine Wettersuche mit einbinden, um die Kombination aus Preisen und warmen Temperaturen für ein potenzielles Ziel zu optimieren, oder eine Hotelsuche, um Ziele mit einer vernünftigen Kombination aus Flug- und Hotelpreisen zu finden. Es gibt im Inter-

e-bol.net

net Tausende von Sites, die Daten zu Reisezielen bereitstellen und die man in eine Optimierung mit einbinden kann.

Die Kayak-API ist auf eine gewisse Zahl von Suchen pro Tag eingeschränkt, aber sie liefert Links zurück, über die jeder Flug oder jedes Hotel direkt gebucht werden kann, was bedeutet, dass Sie die API leicht in eine beliebige Anwendung einbinden können.

Optimierung nach Vorlieben

Sie haben nun ein Beispiel eines Problems gesehen, das per Optimierung gelöst werden kann, aber es gibt viele auf den ersten Blick unabhängige Probleme, die mit den gleichen Methoden angegangen werden können. Denken Sie daran, die wichtigsten Bedingungen für eine Lösung per Optimierung sind eine definierte Zielfunktion und die Tendenz, dass ähnliche Lösungen zu ähnlichen Ergebnissen führen. Nicht jedes Problem mit diesen Eigenschaften wird sich per Optimierung lösen lassen, aber die Chance ist durchaus gegeben, dass man damit interessante Ergebnisse erhält, auf die man selbst nicht gekommen wäre.

Dieser Abschnitt wird ein anderes Problem angehen, das sich für die Optimierung direkt anbietet. Das allgemeine Problem ist, wie man beschränkte Ressourcen an Personen verteilt, die Wünsche geäußert haben, die so gut wie möglich erfüllt werden sollen (oder, abhängig von der Art des Problems, so wenig ärgerlich wie möglich).

Optimierung der Belegung eines Studentenwohnheims

Das Beispielproblem in diesem Abschnitt ist, Studenten auf die Zimmer in einem Wohnheim zu verteilen – wobei möglichst das erste oder zweite Wunschzimmer jedes Studenten berücksichtigt werden soll. Auch wenn dies ein sehr spezielles Beispiel ist, lässt es sich doch leicht auf andere Fälle übertragen – der gleiche Code kann genutzt werden, um Spieler in einem Online-Kartenspiel auf Tische zu verteilen, Bugs in einem großen Entwicklungsprojekt an Entwickler zu verteilen oder sogar um die Hausarbeit auf die Familienmitglieder aufzuteilen. Auch hier ist die Aufgabe wieder, Informationen von den einzelnen Teilnehmern einzusammeln und sie so zu kombinieren, dass ein optimales Ergebnis entsteht.

Es gibt in unserem Beispiel fünf Zimmer, die jeweils Platz für zwei Personen bieten, und zehn Studenten, die hier einziehen wollen. Jeder Student hat ein erstes und ein zweites Wunschzimmer. Erstellen Sie eine neue Datei namens *dorm.py* und fügen Sie die Liste mit Zimmern sowie die Liste mit Personen und ihren Wünschen hinzu:

```
import random
import math

# Die Zimmer mit jeweils zwei Plätzen.
dorms=['Zeus','Athena','Hercules','Bacchus','Pluto']
```

e-bol.net

```
# Die Personen mit ihrer ersten und zweiten Wahl.
prefs=[('Toby', ('Bacchus', 'Hercules')),
       ('Steve', ('Zeus', 'Pluto')),
       ('Andrea', ('Athena', 'Zeus')),
       ('Sarah', ('Zeus', 'Pluto')),
       ('Dave', ('Athena', 'Bacchus')),
       ('Jeff', ('Hercules', 'Pluto')),
       ('Fred', ('Pluto', 'Athena')),
       ('Suzie', ('Bacchus', 'Hercules')),
       ('Laura', ('Bacchus', 'Hercules')),
       ('Neil', ('Hercules', 'Athena'))]
```

Sie sehen auf den ersten Blick, dass nicht jeder Person seine erste Wahl erfüllt werden kann, da es nur zwei Plätze in Bacchus gibt, aber drei Personen, die das Zimmer wollen. Steckt man auch nur eine dieser Personen in das Zimmer ihrer zweiten Wahl, würde das bedeuten, dass in Hercules nicht mehr genug Platz für die Leute wäre, die es eigentlich haben wollen.

Dieses Problem ist klein genug, um es gut im Überblick zu behalten, aber in Wirklichkeit kann es dabei um Hunderte oder Tausende von Studenten gehen, die sich um viel mehr Plätze in einer größeren Auswahl an Zimmern bewerben. Da dieses Beispiel nur etwa 100.000 mögliche Lösungen hat, ist es möglich, sie alle auszutesten und zu sehen, welche die beste ist. Aber die Zahl wächst schnell auf Billionen von Möglichkeiten an, wenn es vier Plätze in jedem Zimmer gibt.

Die Repräsentation der Lösungen ist ein bisschen schwieriger als beim Flug-Problem. Sie könnten theoretisch eine Liste mit Zahlen erstellen – eine für jeden Studenten –, in der jede Zahl für das Zimmer steht, in das der Student gesteckt wird. Das Problem ist dabei, dass die Repräsentation die Lösung nicht darauf beschränkt, dass nur zwei Studenten in jedes Zimmer passen. Eine Liste aus Nullen würde bedeuten, dass jeder in Zeus untergebracht wurde, was keine echte Lösung ist.

Eine Möglichkeit, dies zu beheben, ist das Anpassen der Zielfunktion. Sie könnte für ungültige Lösungen sehr hohe Werte annehmen, aber dann wird es für den Optimierungsalgorithmus sehr schwierig, bessere Lösungen zu finden, da er keine Möglichkeit hat festzustellen, ob er sich nahe an anderen guten oder auch nur gültigen Lösungen befindet. Im Allgemeinen ist es besser, die Prozessorzyklen nicht mit der Suche in ungültigen Lösungen zu belasten.

Eine bessere Möglichkeit besteht darin, einen Weg zu finden, die Lösungen so darzustellen, dass jede gültig ist. Dabei ist eine gültige Lösung nicht notwendigerweise eine gute Lösung – es bedeutet nur, dass genau zwei Studenten jedem Zimmer zugeordnet sind. Man kann zum Beispiel damit arbeiten, dass jedes Zimmer zwei Plätze hat, sodass es in diesem Beispiel insgesamt zehn Plätze gibt. Jeder Student wird dann der Reihe nach einem dieser offenen Plätze zugewiesen – die erste Person kann jedem dieser zehn Plätze zugewiesen werden, die zweite einem der verbleibenden neun Plätze und so weiter.

e-bol.net

Die `domain` für die Suche muss diese Einschränkung widerspiegeln. Fügen Sie diese Zeile der Datei *dorm.py* hinzu:

```
# [(0,9),(0,8),(0,7),(0,6),...,(0,0)]
domain=[(0,(len(dorms)*2)-i-1) for i in range(0,len(dorms)*2)]
```

Der Code, der die Lösung verständlich ausgibt, zeigt, wie die Plätze genutzt werden. Diese Funktion erstellt zuerst eine Liste mit Plätzen – zwei pro Zimmer. Dann läuft sie über jede Zahl in der Lösung und sucht die Raumnummer an dieser Stelle in der Platzliste, wodurch der Raum gefunden wird, dem ein Student zugewiesen wurde. Die Funktion gibt dann den Studenten und den Raum aus und entfernt den Platz aus der Liste, sodass kein anderer Student diesen Platz mehr erhalten kann. Nach der letzten Iteration ist die Platzliste leer, und jeder Student wurde mit seinem Zimmer ausgedruckt. Fügen Sie diese Funktion der Datei *dorm.py* hinzu:

```
def printsolution(vec):
  slots=[]
  # Zwei Plätze für jedes Zimmer.
  for i in range(len(dorms): slots+=[i,i]

  # Schleife über die Studentenzuweisungen.
  for i in range(len(vec)):
    x=int(vec[i])

    # Den Platz aus den verbleibenden wählen.
    dorm=dorms[slots[x]]
    # Student und Zimmer zeigen.
    print prefs[i][0],dorm
    # Platz entfernen.
    del slots[x]
```

Sie können das in Ihre Python-Session importieren und dann versuchen, eine Lösung auszugeben:

```
>>> import dorm
>>> dorm.printsolution([0,0,0,0,0,0,0,0,0,0])
Toby Zeus
Steve Zeus
Andrea Athena
Sarah Athena
Dave Hercules
Jeff Hercules
Fred Bacchus
Suzie Bacchus
Laura Pluto
Neil Pluto
```

Wenn Sie die Zahlen ändern, um andere Lösungen angezeigt zu bekommen, dürfen Sie nicht vergessen, dass jede Zahl im passenden Bereich liegen muss. Das erste Element der Liste kann zwischen 0 und 9 liegen, das zweite zwischen 0 und 8 und so weiter. Wenn eine der Zahlen nicht im entsprechenden Bereich liegt, wird die Funktion eine Ausnahme werfen. Da die Optimierungsfunktion die Zahlen in den

e-bol.net

entsprechenden Bereichen halten wird, die durch den Parameter `domain` vorgegeben wurden, wird dies bei der Optimierung kein Problem sein.

Die Zielfunktion

Die Zielfunktion funktioniert ähnlich wie die Ausgabefunktion. Es wird eine Liste mit Plätzen erstellt, die dann nach und nach wieder entfernt werden, wenn sie gefüllt sind. Die Kosten werden berechnet, indem die aktuelle Zuweisung eines Studenten mit seiner ersten und zweiten Wahl verglichen werden. Sie steigen um 0, wenn der Student seiner ersten Wahl zugewiesen wurde, um 1, wenn es seine zweite Wahl ist, und um 3, wenn er keinen seiner beiden Wunschkandidaten erhalten hat:

```
def dormcost(vec):
  cost=0
  # Erstellen einer Liste mit Plätzen.
  slots=[0,0,1,1,2,2,3,3,4,4]

  # Für jeden Studenten ...
  for i in range(len(vec)):
    x=int(vec[i])
    dorm=dorms[slots[x]]
    pref=prefs[i][1]
    # erste Wahl +0, zweite +1
    if pref[0]==dorm: cost+=0
    elif pref[1]==dorm: cost+=1
    else: cost+=3
    # keine passende Wahl: +3

    # Gewählten Platz entfernen.
    del slots[x]

  return cost
```

Eine nützliche Regel beim Erstellen einer Zielfunktion ist, für die perfekte Lösung (in diesem Beispiel also, dass jeder seine erste Wahl erhält) wenn möglich die Kosten auf null zu setzen. In diesem Fall haben Sie schon festgestellt, dass die perfekte Lösung nicht zu erhalten ist, aber wenn Sie wissen, dass die Kosten dafür null wären, können Sie abschätzen, wie nahe Sie dran sind. Ein weiterer Vorteil dieser Regel ist, dass Sie einen Optimierungsalgorithmus abbrechen können, wenn er schon die perfekte Lösung gefunden hat.

Durchführen der Optimierung

Mit einer Repräsentation für die Lösung, einer Zielfunktion und einer Funktion, die die Ergebnisse ausgibt, haben Sie alles, um die weiter oben definierten Optimierungsfunktionen ausführen zu können. Geben Sie das Folgende in Ihrer Python-Session ein:

e-bol.net

```
>>> reload(dorm)
>>> s=optimization.randomoptimize(dorm.domain,dorm.dormcost)
>>> dorm.dormcost(s)
18
>>> optimization.geneticoptimize(dorm.domain,dorm.dormcost)
13
10
...
4
>>> dorm.printsolution(s)
Toby Athena
Steve Pluto
Andrea Zeus
Sarah Pluto
Dave Hercules
Jeff Hercules
Fred Bacchus
Suzie Bacchus
Laura Athena
Neil Zeus
```

Auch hier können Sie an den Parametern drehen, um zu sehen, ob Sie mit der genetischen Optimierung schneller eine gute Lösung finden.

Netzwerkvisualisierung

Das letzte Beispiel in diesem Kapitel zeigt einen anderen Weg, wie Optimierung bei Problemen verwendet werden kann, die nichts miteinander zu tun haben. In diesem Fall ist das Problem die Visualisierung von Netzwerken. Ein Netzwerk ist in diesem Fall eine beliebige Menge von Dingen, die miteinander verbunden sind. Ein gutes Beispiel im Onlinebereich ist ein Social Network wie MySpace, Facebook oder LinkedIn, in dem Personen miteinander verbunden sind, weil sie Freunde sind oder eine berufliche Beziehung zueinander haben. Jedes Mitglied der Site entscheidet, mit wem es verbunden ist, und gemeinsam bilden sie ein Netzwerk aus Personen. Es ist interessant, solche Netzwerke zu visualisieren, um ihre Struktur zu erkennen, zum Beispiel um Personen zu finden, die Bindeglieder sind (also Personen, die viele Personen kennen oder die als Verbindung zwischen ansonsten nur untereinander befreundeten Cliquen fungieren).

Das Layout-Problem

Wenn man ein Netzwerk zeichnet, um eine große Gruppe von Personen und die Verbindungen zwischen ihnen zu visualisieren, besteht ein Problem darin zu entscheiden, wo jeder Name (oder das Symbol) im Bild untergebracht werden sollte. Schauen Sie sich zum Beispiel das Netzwerk in Abbildung 5-7 an.

e-bol.net

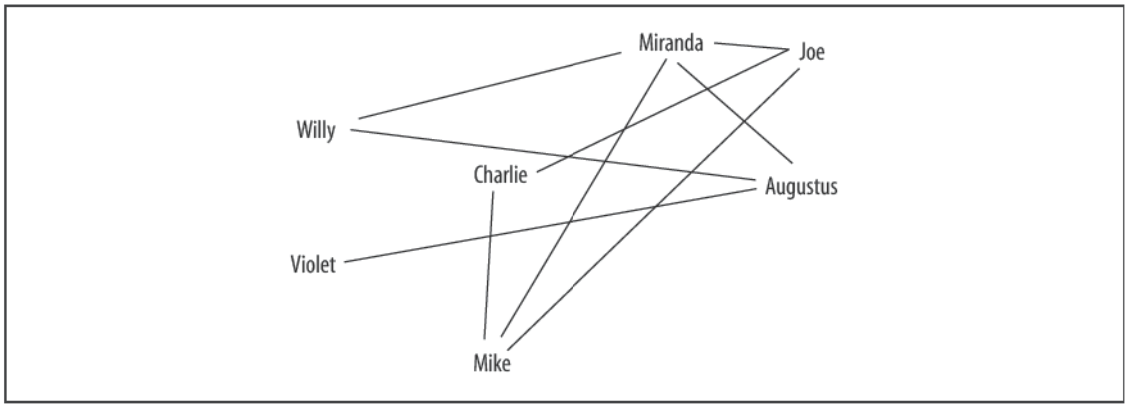

Abbildung 5-7: Ein verwirrendes Netzwerklayout

In dieser Abbildung können Sie sehen, dass Augustus ein Freund von Willy, Violet und Miranda ist. Aber das Layout des Netzwerks ist ein bisschen chaotisch, und wenn man noch mehr Personen hinzufügen würde, wäre es komplett unübersichtlich. Ein deutlich saubereres Layout sehen Sie in Abbildung 5-8.

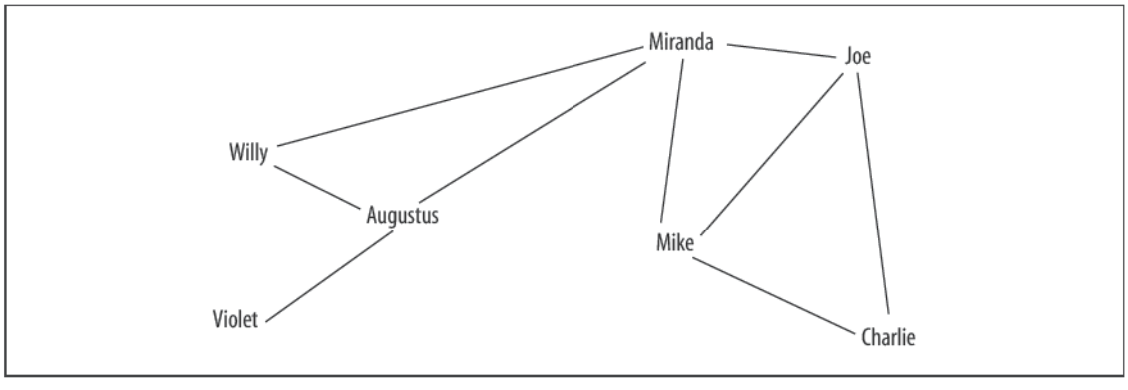

Abbildung 5-8: Ein ordentliches Netzwerklayout

Dieser Abschnitt wird zeigen, wie man mit Optimierungen bessere und weniger verwirrende Zeichnungen erstellen kann. Zunächst legen Sie eine neuen Datei namens *socialnetwork.py* an und fügen ein paar Informationen über einen Unterbereich des Social Network hinzu:

```
import math

people=['Charlie','Augustus','Veruca','Violet','Mike','Joe','Willy','Miranda']

links=[('Augustus', 'Willy'),
       ('Mike', 'Joe'),
       ('Miranda', 'Mike'),
       ('Violet', 'Augustus'),
       ('Miranda', 'Willy'),
       ('Charlie', 'Mike'),
       ('Veruca', 'Joe'),
       ('Miranda', 'Augustus'),
```

e-bol.net

```
('Willy', 'Augustus'),
('Joe', 'Charlie'),
('Veruca', 'Augustus'),
('Miranda', 'Joe')]
```

Das Ziel ist hier, ein Programm zu erstellen, das eine Liste von Informationen über Freundschaften erhält und dann ein leicht zu interpretierendes Netzwerkdiagramm zeichnet. Das wird im Allgemeinen mit einem *Mass-and-Spring*-Algorithmus durchgeführt. Diese Art von Algorithmus hat einen physikalischen Hintergrund, da die verschiedenen Knoten sich gegenseitig abstoßen, während die Verbindungen versuchen, die Knoten näher zusammenzubringen. Damit nimmt das Netzwerk langsam einen Zustand an, in dem die nicht verbundenen Knoten weggeschoben wurden und die verbundenen Knoten zueinander gewandert sind – aber nicht zu nahe.

Leider hält der Mass-and-Spring-Algorithmus die Verbindungen nicht davon ab, sich zu kreuzen. In einem Netzwerk mit vielen Verbindungen wird es dadurch schwierig zu erkennen, welche Knoten miteinander verbunden sind, da die Linien kaum noch gut getrennt werden können. Wenn Sie aber das Layout per Optimierung erstellen, müssen Sie nur eine Zielfunktion aufstellen und dann versuchen, sie zu minimieren. In diesem Fall ist eine interessante Zielfunktion, die Anzahl der Linien zu zählen, die sich kreuzen.

Sich kreuzende Linien zählen

Um die gleichen Optimierungsfunktionen verwenden zu können, die wir schon weiter oben definiert haben, ist es notwendig, eine Lösung als Liste von Zahlen zu repräsentieren. Glücklicherweise lässt sich dieses spezielle Problem ganz einfach so darstellen: Jeder Knoten hat eine x- und eine y-Koordinate, daher können die Koordinaten aller Knoten in eine lange Liste gesteckt werden:

```
sol=[120,200,250,125 ...
```

In dieser Lösung wird Charlie bei (120,200) platziert, Augustus bei (250,125) und so weiter.

Nun wird die neue Zielfunktion einfach die Anzahl der Zeilen zählen, die sich kreuzen. Das Bestimmen einer Formel, die ermittelt, ob sich zwei Linien kreuzen, geht etwas über den Rahmen dieses Kapitels hinaus, aber die prinzipielle Idee ist, den Abschnitt der Linie zu berechnen, an der sie gekreuzt wird. Wenn dieser Abschnitt bei beiden Linien zwischen 0 (einem Ende der Linie) und 1 (dem anderen Ende) liegt, kreuzen sie einander. Wenn der Abschnitt nicht zwischen 0 und 1 liegt, kreuzen sie sich nicht.

Diese Funktion durchläuft jedes Linienpaar und nutzt die aktuellen Koordinaten ihrer Endpunkte, um zu bestimmen, ob sie sich kreuzen. Wenn sie es tun, addiert die Funktion 1 zur Gesamtsumme hinzu. Fügen Sie die Funktion `crosscount` *socialnetwork.py* hinzu:

e-bol.net

```
def crosscount(v):
  # Umwandeln der Zahlenliste in ein Verzeichnis von Personen: (x,y).
  loc=dict([(people[i],(v[i*2],v[i*2+1])) for i in range(0,len(people))])
  total=0

  # Jedes Verbindungspaar durchlaufen.
  for i in range(len(links)):
    for j in range(i+1,len(links)):

      # Positionen ermitteln.
      (x1,y1),(x2,y2)=loc[links[i][0]],loc[links[i][1]]
      (x3,y3),(x4,y4)=loc[links[j][0]],loc[links[j][1]]

      den=(y4-y3)*(x2-x1)-(x4-x3)*(y2-y1)

      # den==0, wenn die Linien parallel sind.
      if den==0: continue

      # Ansonsten sind ua und ub die Abschnitte der Linien,
      # an der sie sich kreuzen.
      ua=((x4-x3)*(y1-y3)-(y4-y3)*(x1-x3))/den
      ub=((x2-x1)*(y1-y3)-(y2-y1)*(x1-x3))/den

      # Wenn der Abschnitt für beide Linien zwischen 0 und 1 ist,
      # kreuzen sie sich.
      if ua>0 and ua<1 and ub>0 and ub<1:
        total+=1
  return total
```

Die `domain` für diese Suche ist der Bereich jeder Koordinate. In diesem Beispiel können Sie davon ausgehen, dass das Netzwerk in einem 400 × 400-Pixel-Bild visualisiert werden wird, daher wird die Domain ein wenig kleiner sein, um einen Rahmen zu ermöglichen. Fügen Sie diese Zeilen am Ende von *socialnetwork.py* ein:

```
domain=[(10,370)]*(len(people)*2)
```

Jetzt können Sie versuchen, einige der Optimierungsalgorithmen auszuführen, um eine Lösung zu finden, in der sich nur sehr wenige Linien kreuzen. Importieren Sie *socialnetwork.py* in Ihre Python-Session und testen Sie ein paar der Optimierungsalgorithmen aus:

```
>>> import socialnetwork
>>> import optimization
>>> sol=optimization.randomoptimize(socialnetwork.domain,socialnetwork.crosscount)
>>> socialnetwork.crosscount(sol)
12
>>> sol=optimization.annealingoptimize(socialnetwork.domain,
    socialnetwork.crosscount,step=50,cool=0.99)
>>> socialnetwork.crosscount(sol)
1
>>> sol
[324, 190, 241, 329, 298, 237, 117, 181, 88, 106, 56, 10, 296, 370, 11, 312]
```

e-bol.net

Die simulierte Abkühlung scheint eine Lösung zu finden, in der sich nur sehr wenige Linien kreuzen, aber die Liste der Koordinaten lässt sich nur schwer interpretieren. Der nächste Abschnitt wird Ihnen zeigen, wie Sie das Netzwerk automatisch zeichnen lassen können.

Das Netzwerk zeichnen

Sie werden wieder die Python Imaging Library brauchen, die wir schon in Kapitel 3 genutzt haben. Wenn Sie sie noch nicht installiert haben, werfen Sie bitte einen Blick in Anhang A. Dort gibt es Informationen darüber, wie Sie die neueste Version erhalten und in Ihrer Python-Instanz installieren können.

Der Code zum Zeichnen des Netzwerks ist nicht sehr knifflig. Er erstellt nur ein Bild, zeichnet die Verbindungen zwischen den verschiedenen Personen und dann die Knoten für die Personen. Die Namen werden danach gezeichnet, damit die Linien sie nicht verdecken. Fügen Sie diese Funktion der Datei *socialnetwork.py* hinzu:

```
def drawnetwork(sol):
  # Erstellen des Bildes.
  img=Image.new('RGB',(400,400),(255,255,255))
  draw=ImageDraw.Draw(img)

  # Erstellen des Positions-Dictionary.
  pos=dict([(people[i],(sol[i*2],sol[i*2+1])) for i in range(0,len(people))])

  # Verbindungen zeichnen.
  for (a,b) in links:
    draw.line((pos[a],pos[b]),fill=(255,0,0))

  # Personen zeichnen.
  for n,p in pos.items( ):
    draw.text(p,n,(0,0,0))

  img.show( )
```

Um diese Funktion in Ihrer Python-Session auszuführen, laden Sie einfach das Modul neu und rufen die Funktion für Ihre Lösung auf:

```
>>> reload(socialnetwork)
>>> drawnetwork(sol)
```

Abbildung 5-9 zeigt ein mögliches Ergebnis der Optimierung.

Natürlich wird Ihre Lösung anders aussehen. Manchmal wird die Lösung ziemlich verrückt erscheinen – da das Ziel nur ist, die Anzahl der sich kreuzenden Linien zu minimieren, wird die Zielfunktion keine Strafpunkte dafür vergeben, wenn zwischen den Linien sehr kleine Winkel liegen oder sich zwei Knoten nahe beieinander befinden. In dieser Hinsicht ist die Optimierung vergleichbar mit einem Genie, das Ihre Wünsche wortwörtlich ausführt, daher ist es immer wichtig, sich darüber im

e-bol.net

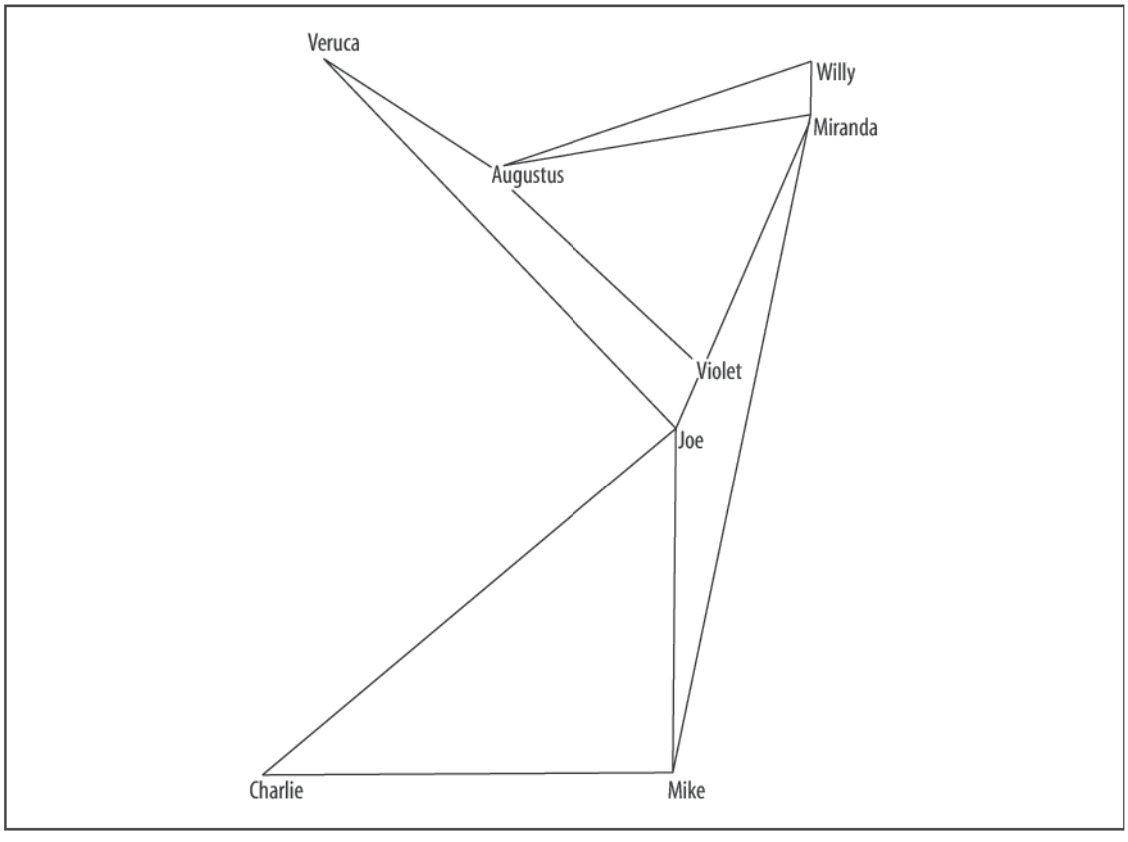

Abbildung 5-9: Layout, das aus einer Optimierung für möglichst wenige sich kreuzende Linien hervorgegangen ist

Klaren zu sein, was Sie eigentlich wollen. Oft ergibt sich eine Lösung, die die ursprünglichen Kriterien von »gut« erfüllt, aber überhaupt nicht so aussieht, wie Sie es sich vorgestellt haben.

Eine einfache Möglichkeit, eine Lösung abzustrafen, die zwei Knoten zu dicht beieinanderliegen lässt, ist das Berechnen des Abstands zwischen den Knoten und dem Teilen durch eine gewünschte Minimaldistanz. Sie können diesen Code am Ende von `crosscount` (vor der Anweisung `return`) hinzufügen, um eine zusätzliche Strafe zu ermöglichen.

```
for i in range(len(people)):
  for j in range(i+1,len(people)):
    # Positionen der beiden Knoten ermitteln.
    (x1,y1),(x2,y2)=loc[people[i]],loc[people[j]]

    # Abstand bestimmen.
    dist=math.sqrt(math.pow(x1-x2,2)+math.pow(y1-y2,2))
    # Strafe für Knoten, die näher als 50 Pixel beieinanderliegen.
    if dist<50:
      total+=(1.0-(dist/50.0))
```

Damit entstehen höhere Kosten für jedes Knotenpaar, das weniger als 50 Pixel auseinanderliegt – abhängig davon, wie nahe sie beieinander sind. Wenn sie sich an

e-bol.net

genau demselben Platz befinden, ist die Strafe 1. Führen Sie die Optimierung erneut aus, um zu sehen, ob diese Ergebnisse ein übersichtlicheres Layout liefern.

Andere Möglichkeiten

Dieses Kapitel hat drei vollständig unterschiedliche Anwendungen für Optimierungsalgorithmen vorgestellt, aber das ist nur ein kleiner Teil dessen, was möglich ist. Wie schon im Laufe des Kapitels erwähnt, sind die entscheidenden Schritte, eine Repräsentation und eine Zielfunktion festzulegen. Wenn Sie das hinbekommen, ist die Chance durchaus gegeben, dass Sie mithilfe der Optimierung Lösungen für Ihr Problem finden.

Eine interessante Aufgabe könnte sein, eine große Gruppe von Personen zu nehmen und sie in Teams aufzuteilen, in denen die Fähigkeiten der Mitglieder gleich verteilt sind. In einem Quiz kann es wünschenswert sein, Teams so zusammenzustellen, dass jedes Team ein in etwa vergleichbares Wissen in den Bereichen Sport, Geschichte, Literatur und Fernsehen besitzt. Eine andere Möglichkeit besteht darin, Aufgaben in Projekten zuzuweisen, indem man die Fähigkeiten der einzelnen Personen in Betracht zieht. Die Optimierung kann den besten Weg präsentieren, die Aufgaben aufzuteilen, sodass die Aufgabenliste in möglichst kurzer Zeit abgearbeitet ist.

Hat man eine lange Liste von Websites, die mit Schlüsselwörtern getaggt sind, kann es interessant sein, eine optimale Gruppe von Websites für eine vom Benutzer vorgegebene Menge von Schlüsselwörtern zu liefern. Die optimale Gruppe würde eine Reihe von Websites enthalten, die nicht viele Schlüsselwörter gemeinsam haben, aber so viele vorgegebene Schlüsselwörter wie möglich abdecken.

Übungen

1. *Zielfunktion bei Gruppenreisen*. Fügen Sie die Gesamtflugzeit als Kosten mit 0,50 Dollar pro Minute hinzu. Als Nächstes addieren Sie eine Strafe von 20 Dollar, falls jemand vor 8.00 Uhr morgens am Flughafen eintrudeln muss.
2. *Ausgangspunkte für die Abkühlung*. Das Ergebnis der simulierten Abkühlung hängt stark vom Ausgangspunkt ab. Erstellen Sie eine neue Optimierungsfunktion, die die simulierte Abkühlung von mehreren Ausgangspunkten aus durchführt und das beste Ergebnis zurückliefert.
3. *Abbruchkriterien für die genetische Optimierung*. Eine Funktion ruft in diesem Kapitel die genetische Optimierung mit einer vorgegebenen Anzahl von Iterationen auf. Ändern Sie sie so ab, dass sie abbricht, wenn es seit zehn Iterationen keine Verbesserung der besten Lösung mehr gibt.

e-bol.net

4. *Preise für Hin- und Rückflüge*. Die Funktion, die die Flugdaten bei Kayak zusammensucht, schaut bisher nur nach einfachen Flügen. Die Preise sind aber wahrscheinlich geringer, wenn man Hin- und Rückflug zusammen bucht. Passen Sie den Code an, um die Preise für den Hin- und Rückflug zusammen zu ermitteln, und ändern Sie die Zielfunktion, sodass sie die Preise für bestimmte Kombinationen aus Hin- und Rückflügen nachschlägt, anstatt einfach die Einzelpreise aufzusummieren.
5. *Studenten einteilen*. Stellen Sie sich vor, die Studenten wollten keine Wünsche bezüglich der Räume, sondern ihrer Partner abgeben. Wie würden Sie dann Lösungen repräsentieren, um die Studenten in Paare aufzuteilen? Wie würde die Zielfunktion aussehen?
6. *Strafen bei Linienwinkeln*. Fügen Sie dem Algorithmus für das Netzwerklayout zusätzliche Kosten hinzu, wenn der Winkel zwischen zwei Linien, die von der gleichen Person ausgehen, sehr klein ist. (Hinweis: Sie können das *Vektor-Kreuz-Produkt* nutzen.)

e-bol.net

KAPITEL 6

Dokumente filtern

Dieses Kapitel wird zeigen, wie man Dokumente anhand ihres Inhalts klassifizieren kann – eine sehr praktische Anwendung der künstlichen Intelligenz und eine, die auch immer mehr Verbreitung findet. Die vielleicht nützlichste und bekannteste Anwendung der Dokumentenfilterung ist das Ausmerzen von Spam. Ein großes Problem durch die weit verbreitete Nutzung von E-Mail und die extrem geringen Kosten beim Verschicken ist, dass jeder, dessen Adresse in die falschen Hände gerät, sehr wahrscheinlich unerwünschte kommerzielle E-Mails erhält, wodurch es schwierig wird, die Nachrichten herauszupicken, die tatsächlich interessant sind.

Das Spam-Problem tritt allerdings nicht nur bei E-Mails auf. Websites wurden mit der Zeit immer interaktiver und ermöglichen den Benutzern, Kommentare abzugeben oder auch selbst Inhalte zu erstellen, was das Spam-Problem noch verschlimmert hat. Öffentliche Boards, wie zum Beispiel Yahoo! Groups und das Usenet sind schon lange Opfer von Postings, die keine Beziehung zum Thema des Boards haben oder dubiose Produkte anpreisen. Blogs und Wikis sehen sich nun dem gleichen Problem gegenüber. Wenn Sie eine Anwendung bauen, die es jedem ermöglicht, Beiträge zu leisten, sollten Sie auch immer eine Strategie für den Umgang mit Spam parat haben.

Die Algorithmen, die in diesem Kapitel beschrieben werden, sind nicht speziell auf den Umgang mit Spam ausgelegt. Da sie ein allgemeineres Problem lösen, nämlich zu lernen, ob ein Dokument in eine Kategorie oder in eine andere gehört, können sie auch für weniger unangenehme Zwecke genutzt werden. Ein Beispiel könnte das automatische Aufteilen Ihrer Inbox in private und arbeitsbezogene E-Mails sein – in Abhängigkeit vom Inhalt der Nachrichten. Eine andere Möglichkeit besteht darin, E-Mail-Nachrichten zu erkennen, die Informationen anfordern, und sie automatisch an die dafür kompetenteste Person weiterzuleiten, die darauf reagieren kann. Das Beispiel am Ende dieses Kapitels wird automatische Beiträge aus einem RSS-Feed in verschiedene Kategorien einteilen.

e-bol.net

Spam filtern

Frühe Versuche, Spam zu filtern, griffen alle auf regelbasierte Klassifizierer zurück, bei denen eine Person eine Reihe von Regeln entwarf, die festlegen sollten, ob eine Nachricht Spam war oder nicht. Solche Regeln beinhalteten im Allgemeinen Dinge wie eine zu üppige Verwendung von Großbuchstaben, Wörter, die mit pharmazeutischen Produkten in Zusammenhang stehen, oder besonders knallige HTML-Farben. Das Problem bei regelbasierten Klassifizierern wurde schnell deutlich – die Spammer machten die Regeln ausfindig und verwendeten in ihren Mails nicht mehr die offensichtlichsten Verhaltensmuster, um die Filter umgehen zu können, während andere, die nie die korrekte Verwendung der Capslock-Taste beigebracht bekommen hatten, ihre guten E-Mails als Spam klassifiziert vorfanden.

Das andere Problem bei regelbasierten Filtern ist, dass eine Einstufung als Spam sehr davon abhängt, wo eine Mail gepostet wurde und für wen sie entstand. Schlüsselwörter, die für einen bestimmten Benutzer, ein Board oder ein Wiki eindeutig auf Spam hinweisen, können für andere ganz normal sein. Um dieses Problem zu lösen, wird dieses Kapitel Programme zeigen, die *lernen*, was Spam ist – indem Sie ihnen mitteilen, welche E-Mails Spam sind und welche nicht. Das geschieht einmal zu Beginn, aber auch später, während Sie weitere Mails erhalten. Dadurch können Sie viele Instanzen und Datenmengen für einzelne Benutzer, Gruppen oder Sites erstellen, die alle ihre eigenen Ideen dazu entwickeln, was Spam ist und was nicht.

Dokumente und Wörter

Der Klassifizierer, den Sie erstellen werden, braucht *Merkmale*, die Sie zum Klassifizieren von verschiedenen Elementen nutzen. Ein Merkmal ist alles, was Sie in einem Element entweder als vorhanden oder als nicht vorhanden feststellen können. Wenn Sie Dokumente klassifizieren wollen, sind die Dokumente die Elemente und die Merkmale die Wörter in den Dokumenten. Wenn man Wörter als Merkmale verwendet, geht man davon aus, dass manche Wörter häufiger in Spam-Texten als in normalen Texten auftauchen. Auf dieser grundlegenden Voraussetzung bauen die meisten Spam-Filter auf. Merkmale müssen allerdings keine einzelnen Wörter sein, es kann sich auch um Wortpaare, Phrasen oder irgendetwas anderes handeln, das in einem bestimmten Dokument als vorhanden oder nicht vorhanden klassifiziert werden kann.

Erstellen Sie eine neue Datei namens *docclass.py* und fügen Sie eine Funktion `getwords` hinzu, um die Merkmale aus dem Text zu extrahieren:

```
import re
import math

def getwords(doc):
  splitter=re.compile('\\W*')
```

e-bol.net

```
    # Aufteilen der Wörter an Zeichen, die keine Buchstaben sind.
    words=[s.lower( ) for s in splitter.split(doc)
            if len(s)>2 and len(s)<20]

    # Jedes Wort einmal zurückgeben.
    return dict([(w,1) for w in words])
```

Diese Funktion teilt den Text in Wörter auf, indem sie sie bei jedem Zeichen trennt, das kein Buchstabe ist. Damit verbleiben nur die eigentlichen Wörter, die alle in Kleinbuchstaben umgewandelt werden.

Das Ermitteln der zu verwendenden Merkmale ist ebenso knifflig wie wichtig. Die Merkmale müssen so allgemein sein, dass sie häufig genug auftauchen, aber nicht so häufig, dass sie in jedem einzelnen Dokument vorkommen. Theoretisch könnte der gesamte Text des Dokuments ein Merkmal sein, aber das wäre mit fast hundertprozentiger Sicherheit nutzlos, solange Sie nicht immer und immer wieder genau die gleiche Nachricht erhalten. Das andere Extrem wäre die Definition einzelner Buchstaben als Merkmale, aber da die meisten Buchstaben in so gut wie jeder E-Mail-Nachricht auftauchen, würde man mit ihnen die erwünschten von den unerwünschten Nachrichten nur sehr schlecht trennen können. Selbst die Wahl von Wörtern als Merkmale sorgt für weitere Fragen: Wie genau sollen die Wörter getrennt werden? Sollen Satzzeichen dazugehören? Welche Header-Informationen sollen mit berücksichtigt werden?

Wenn man über die Bestimmung als Merkmal entscheidet, muss man auch berücksichtigen, wie gut sie die Dokumente in die verschiedenen Zielkategorien unterteilen. Der obige Code von `getwords` zum Beispiel verringert die Gesamtmenge der möglichen Merkmale, indem er sie in Kleinbuchstaben umwandelt. Er erkennt also, dass ein Wort am Satzanfang mit einem Großbuchstaben zu Beginn das Gleiche ist wie das Wort in Kleinbuchstaben mitten im Satz – eine gute Sache, da Wörter mit unterschiedlicher Groß- und Kleinschreibung fast immer das Gleiche bedeuten. Andererseits geht einem durch diese Funktion die ganze Problematik des SCHREIENs durch die Lappen, die in vielen Spam-Mails genutzt wird, was eventuell entscheidend sein kann, wenn man die Dokumente unterteilen möchte. Eine andere Alternative könnte sein, ein Merkmal zu haben, das dann vorhanden ist, wenn mehr als die Hälfte der Wörter komplett in Großbuchstaben geschrieben wurde.

Wie Sie sehen können, gehören zur Auswahl der möglichen Merkmale viele Überlegungen, und man kann endlos viele Anpassungen vornehmen. Hier können Sie erst einmal die einfache Funktion `getwords` nutzen. Weiter unten im Kapitel werden Sie ein paar Ideen erhalten, wie man die Extraktion der Merkmale noch verbessern kann.

e-bol.net

Trainieren des Klassifizierers

Die in diesem Kapitel behandelten Klassifizierer lernen das Klassifizieren eines Dokuments, indem sie trainiert werden. Viele der anderen Algorithmen in diesem Buch, wie zum Beispiel das neuronale Netz, das Sie in Kapitel 4 kennengelernt haben, lernen, indem sie Beispiele für richtige Antworten einlesen. Je mehr Beispiele für Dokumente und ihre korrekten Klassifikationen sie zu sehen bekommen, desto bessere Einstufungen nimmt der Klassifizierer vor. Er ist zudem extra so ausgelegt, dass er erst einmal sehr unsicher beginnt, dann aber seine Klassifizierungssicherheit wächst, während er lernt, welche Merkmale für eine Entscheidung wichtig sind.

Das Erste, was Sie brauchen, ist nun eine Klasse, die den Klassifizierer repräsentiert. Diese Klasse wird kapseln, was der Klassifizierer bisher gelernt hat. Der Vorteil einer Strukturierung des Moduls auf diese Art und Weise ist, dass Sie mehrere Klassifizierer für verschiedene Benutzer, Gruppen oder Abfragen instanziieren und sie unterschiedlich trainieren können, um auf die Bedürfnisse einzelner Gruppen eingehen zu können. Erstellen Sie eine Klasse namens `classifier` in *docclass.py*:

```
class classifier:
  def __init__(self,getfeatures,filename=None):
    # Zähler der Merkmal-Kategorie-Kombinationen
    self.fc={}
    # Zähler der Dokumente in jeder Kategorie
    self.cc={}
    self.getfeatures=getfeatures
```

Die drei Instanzvariablen sind `fc`, `cc` und `getfeatures`. Die Variable `fc` wird die Summen für die verschiedenen Merkmale in den unterschiedlichen Klassifikationen zählen, zum Beispiel:

```
{'python': {'bad': 0, 'good': 6}, 'the': {'bad': 3, 'good': 3}}
```

Das gibt an, dass das Wort »the« dreimal in Dokumenten aufgetaucht ist, die als schlecht klassifiziert wurden, und ebenso dreimal in »guten« Dokumenten. Das Wort »Python« ist bisher nur in guten Dokumenten erschienen.

Die Variable `cc` ist ein Dictionary, in dem vermerkt ist, wie oft jede Klassifikation genutzt wurde. Das wird benötigt, um die Wahrscheinlichkeitsberechnungen durchführen zu können, die wir gleich noch besprechen werden. Die letzte Instanzvariable `getfeatures` ist die Funktion, die genutzt wird, um die Merkmale aus den zu klassifizierenden Elementen zu extrahieren – in diesem Beispiel also die weiter oben definierte Funktion `getwords`.

Die Methoden in der Klasse werden die Dictionaries nicht direkt verwenden, da man sich damit mögliche Optionen zum Speichern der Trainingsdaten in einer Datei oder Datenbank verbauen würde. Erstellen Sie daher diese Hilfsfunktionen, um die Zähler zu erhöhen und auszulesen:

e-bol.net

```
  # Zähler für ein Merkmals-Kategorie-Paar erhöhen.
  def incf(self,f,cat):
    self.fc.setdefault(f,{})
    self.fc[f].setdefault(cat,0)
    self.fc[f][cat]+=1

  # Zähler für eine Kategorie erhöhen.
  def incc(self,cat):
    self.cc.setdefault(cat,0)
    self.cc[cat]+=1

  # Häufigkeit, mit der ein Merkmal in einer Kategorie erschienen ist.
  def fcount(self,f,cat):
    if f in self.fc and cat in self.fc[f]:
      return float(self.fc[f][cat])
    return 0.0

  # Anzahl der Elemente in einer Kategorie.
  def catcount(self,cat):
    if cat in self.cc:
      return float(self.cc[cat])
    return 0

  # Gesamtanzahl an Elementen.
  def totalcount(self):
    return sum(self.cc.values( ))

  # Liste aller Kategorien.
  def categories(self):
    return self.cc.keys( )
```

Die Methode `train` erwartet ein Element (in diesem Fall ein Dokument) und eine Klassifikation. Sie verwendet die Klassenfunktion `getfeatures`, um das Element in seine einzelnen Merkmale aufzuteilen. Dann ruft sie `incf` auf, um die Zähler dieser Klassifikation für jedes Element zu erhöhen. Schließlich erhöht sie noch die Gesamtanzahl für diese Klassifikation:

```
  def train(self,item,cat):
    features=self.getfeatures(item)
    # Zähler für jedes Merkmal in dieser Kategorie erhöhen.
    for f in features:
      self.incf(f,cat)

    # Zähler für diese Kategorie erhöhen.
    self.incc(cat)
```

Sie können prüfen, ob Ihre Klasse ordentlich funktioniert, indem Sie eine neue Python-Session starten und dieses Modul importieren:

```
$ python
>>> import docclass
>>> cl=docclass.classifier(docclass.getwords)
>>> cl.train('the quick brown fox jumps over the lazy dog','good')
```

e-bol.net

```
>>> cl.train('make quick money in the online casino','bad')
>>> cl.fcount('quick','good')
1.0
>>> cl.fcount('quick','bad')
1.0
```

An dieser Stelle ist es nützlich, eine Funktion zu haben, die ein paar Beispiel-Trainingsdaten in den Klassifizierer schiebt, damit Sie sie nicht jedes Mal per Hand trainieren müssen, wenn Sie sie anlegen. Fügen Sie diese Funktion am Anfang von *docclass.py* hinzu:

```
def sampletrain(cl):
  cl.train('Nobody owns the water.','good')
  cl.train('the quick rabbit jumps fences','good')
  cl.train('buy pharmaceuticals now','bad')
  cl.train('make quick money at the online casino','bad')
  cl.train('the quick brown fox jumps','good')
```

Wahrscheinlichkeiten berechnen

Sie haben nun Zähler, die angeben, wie oft eine E-Mail in jeder Kategorie auftaucht, daher ist der nächste Schritt, diese Zahlen in Wahrscheinlichkeiten umzurechnen. Eine Wahrscheinlichkeit ist eine Zahl zwischen 0 und 1, die angibt, wie wahrscheinlich ein Ereignis ist. In diesem Fall können Sie die Wahrscheinlichkeit berechnen, dass sich ein Wort in einer bestimmten Kategorie befindet, indem Sie die Häufigkeit des Auftretens dieses Worts in einem Dokument in dieser Kategorie durch die Gesamtanzahl an Dokumenten in dieser Kategorie teilen.

Fügen Sie eine Methode namens `fprob` zur Klasse `classifier` hinzu:

```
def fprob(self,f,cat):
  if self.catcount(cat)==0: return 0

  # Gesamthäufigkeit dieses Merkmals in dieser Kategorie, geteilt durch
  # die Gesamtanzahl der Elemente in dieser Kategorie.
  return self.fcount(f,cat)/self.catcount(cat)
```

Dies wird als *bedingte Wahrscheinlichkeit* bezeichnet, normalerweise als *Pr(A | B)* geschrieben und gesprochen als »die Wahrscheinlichkeit für A unter der Voraussetzung von B«. In diesem Beispiel ist die Zahl, die Sie jetzt haben, *Pr(Wort | Klassifikation)*, also berechnen Sie für eine gegebene Klassifikation die Wahrscheinlichkeit, dass ein bestimmtes Wort auftaucht.

Sie können diese Funktion in Ihrer Python-Session ausprobieren:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.py'>
>>> cl=docclass.classifier(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> cl.fprob('quick','good')
0.66666666666666663
```

e-bol.net

Sie sehen, dass das Wort »quick« in zwei der drei Dokumente auftaucht, die als gut klassifiziert wurden, was bedeutet, dass es eine Wahrscheinlichkeit von *Pr(quick | good)* = 0,666 (eine 2/3-Chance) gibt, dass ein gutes Dokument dieses Wort enthalten wird.

Mit einer vernünftigen Schätzung beginnen

Die Methode `fprob` liefert ein genaues Ergebnis für die Merkmale und Klassifikationen, die sie bisher kennengelernt hat, aber sie hat ein kleines Problem – nutzt sie nur die Informationen, die sie bisher zu Gesicht bekam, ist sie während der ersten Trainingssessions ausgesprochen empfindlich und reagiert sehr stark auf Wörter, die selten vorkommen. In den Beispiel-Trainingsdaten taucht das Wort »money« nur in einem Dokument auf und wird als schlecht klassifiziert, da es sich um eine Casinowerbung handelt. Da das Wort »money« in einem schlechten und keinem guten Dokument vorkommt, ist die mit `fprob` ermittelte Wahrscheinlichkeit, dass es in der guten Kategorie vorkommt, gleich null. Das ist ein bisschen extrem, da »money« ein ziemlich neutrales Wort sein kann, das nur zufällig das erste Mal in einem schlechten Dokument auftauchte. Es wäre für den Wert deutlich realistischer, wenn er sich nach und nach null nähern würde, wenn das Wort in mehr und mehr Dokumenten der gleichen Kategorie gefunden wird.

Um das zu erreichen, werden Sie eine *angenommene Wahrscheinlichkeit* nutzen müssen, die Sie verwenden, wenn Sie nur sehr wenige Informationen über das fragliche Merkmal haben. Ein guter Ausgangspunkt dafür ist 0,5. Sie werden auch entscheiden müssen, wie stark die angenommene Wahrscheinlichkeit *gewichtet* werden soll – eine Gewichtung von 1 bedeutet, dass sie genauso viel gewichtet wird wie ein Wort. Die gewichtete Wahrscheinlichkeit liefert einen gewichteten Durchschnitt von `getprobability` und der angenommenen Wahrscheinlichkeit zurück.

Im Beispiel von »money« ist die gewichtete Wahrscheinlichkeit für das Wort »money« für alle Kategorien zunächst 0,5. Nachdem der Klassifizierer mit einem schlechten Dokument trainiert wurde und davon ausgeht, dass »money« in diese schlechte Kategorie passt, wird die Wahrscheinlichkeit 0,75, dass das Wort schlecht ist. Das liegt an Folgendem:

```
(weight*assumedprob + count*fprob)/(count+weight)
= (1*1.0+1*0.5)/(1.0 + 1.0)
= 0.75
```

Fügen Sie die Methode `weightedprob` zu Ihrer Klasse `classifier` hinzu:

```
def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
  # Aktuelle Wahrscheinlichkeit berechnen.
  basicprob=prf(f,cat)

  # Zählen, wie oft dieses Merkmal in allen Kategorien vorkommt.
  totals=sum([self.fcount(f,c) for c in self.categories()])
```

e-bol.net

```
    # Den gewichteten Durchschnitt berechnen.
    bp=((weight*ap)+(totals*basicprob))/(weight+totals)
    return bp
```

Sie können die Funktion jetzt in Ihrer Python-Session ausprobieren. Laden Sie das Modul neu und führen Sie die Methode `sampletrain` nochmals aus, da durch das Erstellen einer neuen Instanz der Klasse die bestehenden Trainingsdaten gelöscht sind:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.pyc'>
>>> cl=docclass.classifier(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> cl.weightedprob('money','good',cl.fprob)
0.25
>>> docclass.sampletrain(cl)
>>> cl.weightedprob('money','good',cl.fprob)
0.16666666666666666
```

Wie Sie sehen können, sorgt ein erneutes Ausführen der Methode `sampletrain` dafür, dass sich der Klassifizierer bei den verschiedenen Wortwahrscheinlichkeiten noch sicherer ist, da sie stärker von ihren angenommenen Wahrscheinlichkeiten wegdriften.

Die angenommene Wahrscheinlichkeit von 0,5 wurde einfach deshalb gewählt, weil sie genau auf dem halben Weg zwischen 0 und 1 liegt. Es ist aber möglich, dass Sie bessere Hintergrundinformationen haben als ein komplett untrainierter Klassifizierer. So kann eine Person, die damit beginnt, einen Spam-Filter zu trainieren, zum Beispiel die Wahrscheinlichkeiten von den schon trainierten Spam-Filtern anderer Personen als angenommene Wahrscheinlichkeiten verwenden. Der Benutzer hat immer noch einen Spam-Filter, der für ihn angepasst ist, aber er kann schon besser mit Wörtern umgehen, die selten auftauchen.

Ein naiver Klassifizierer

Nachdem Sie nun die Wahrscheinlichkeiten für ein Dokument in einer Kategorie mit einem bestimmten Wort ermittelt haben, müssen Sie nun einen Weg finden, die einzelnen Wortwahrscheinlichkeiten zu kombinieren, um die Wahrscheinlichkeit zu ermitteln, dass das gesamte Dokument in eine bestimmte Kategorie passt. Dieses Kapitel wird zwei verschiedene Klassifikationsmethoden vorstellen. Beide funktionieren in den meisten Situationen, aber für bestimmte Aufgaben unterscheidet sich ihre Performance ein wenig. Der in diesem Abschnitt behandelte Klassifizierer wird als *naiver Bayes-Klassifizierer* bezeichnet.

Diese Methode wird als *naiv* bezeichnet, weil sie davon ausgeht, dass die zu kombinierenden Wahrscheinlichkeiten *unabhängig* voneinander sind. Das bedeutet, die Wahrscheinlichkeit, dass sich ein bestimmtes Wort im Dokument in einer

e-bol.net

bestimmten Kategorie befindet, steht in keinem Zusammenhang zur Wahrscheinlichkeit der anderen Wörter in dieser Kategorie. Das ist eigentlich eine falsche Annahme, da Sie vermutlich feststellen werden, dass Dokumente mit dem Wort »Casino« sehr viel häufiger auch das Wort »money« enthalten als Dokumente über Python-Programmierung.

Das bedeutet, dass Sie die Wahrscheinlichkeiten, die durch den naiven Bayes-Klassifizierer erstellt wurden, nicht als tatsächliche Wahrscheinlichkeit dafür nutzen können, dass ein Dokument zu einer Kategorie gehört, da die Annahme der Unabhängigkeit für eine Ungenauigkeit sorgt. Aber Sie können die Ergebnisse für verschiedene Kategorien trotzdem *vergleichen* und schauen, welche die höchste Wahrscheinlichkeit besitzt. Trotz der zugrunde liegenden falschen Annahme hat sich im realen Einsatz gezeigt, dass dies eine erstaunlich effektive Methode zu Klassifizieren von Dokumenten ist.

Wahrscheinlichkeit eines ganzen Dokuments

Um den naiven Bayes-Klassifizierer zu nutzen, müssen Sie zunächst die Wahrscheinlichkeit, dass sich ein ganzes Dokument in einer Klasse befindet, bestimmen. Wie schon erwähnt, werden Sie davon ausgehen, dass die Wahrscheinlichkeiten unabhängig voneinander sind, was bedeutet, dass Sie die Gesamtwahrscheinlichkeit ermitteln können, indem Sie sie miteinander multiplizieren.

Gehen wir zum Beispiel einmal davon aus, dass Sie festgestellt haben, dass das Wort »Python« in 20% Ihrer schlechten Dokumente auftaucht – $Pr(Python \mid Bad) = 0{,}2$ – und dass das Wort »casino« in 80% Ihrer schlechten Dokumente zu finden ist ($Pr(Casino \mid Bad) = 0{,}8$). Damit erwarten Sie, dass die unabhängige Wahrscheinlichkeit für beide Wörter in einem schlechten Dokument – $Pr(Python \,\&\, Casino \mid Bad)$ – den Wert $0{,}8 \times 0{,}2 = 0{,}16$ besitzt. Daraus können Sie ersehen, dass es beim Berechnen der Dokumentwahrscheinlichkeit nur um das Multiplizieren aller Wahrscheinlichkeiten der einzelnen Wörter in diesem Dokument geht.

Erstellen Sie in *docclass.py* eine neue Unterklasse von `classifier` namens `naivebayes` und legen Sie eine Methode `docprob` an, die die Merkmale (Wörter) extrahiert und alle ihre Wahrscheinlichkeiten miteinander multipliziert, um eine Gesamtwahrscheinlichkeit zu erhalten:

```
class naivebayes(classifier):
  def docprob(self,item,cat):
    features=self.getfeatures(item)

    # Multiplizieren der Wahrscheinlichkeiten aller Merkmale.
    p=1
    for f in features: p*=self.weightedprob(f,cat,self.fprob)
    return p
```

e-bol.net

Sie wissen nun, wie Sie *Pr(Dokument | Kategorie)* berechnen, aber das allein ist noch nicht sehr hilfreich. Um Dokumente zu klassifizieren, brauchen Sie eigentlich *Pr(Kategorie | Dokument)*. Mit anderen Worten: Wie groß ist die Wahrscheinlichkeit für ein *gegebenes* Dokument, dass es in diese Kategorie passt? Glücklicherweise hat ein britischer Mathematiker namens Thomas Bayes vor ungefähr 250 Jahren herausgefunden, wie man das herausbekommt.

Eine kurze Einführung in das Bayes-Theorem

Das Bayes-Theorem bietet die Möglichkeit, bedingte Wahrscheinlichkeiten zu vertauschen. Normalerweise schreibt man es so:

Pr(A | B) = Pr(B | A) x Pr(A)/Pr(B)

Im Beispiel wird das zu:

Pr(Kategorie | Dokument) = Pr(Dokument | Kategorie) x Pr(Kategorie) / Pr(Dokument)

Der vorige Abschnitt hat gezeigt, wie man *Pr(Dokument | Kategorie)* berechnen kann, aber wie sieht es mit den beiden anderen Werten in der Gleichung aus? Nun, *Pr(Kategorie)* ist die Wahrscheinlichkeit, dass sich ein zufällig ausgewähltes Dokument in dieser Kategorie befindet, daher ist der Wert einfach die Anzahl der Dokumente in dieser Kategorie geteilt durch die Gesamtanzahl an Dokumenten.

Den Wert von *Pr(Dokument)* könnten Sie berechnen, aber das wäre ein unnötiger Aufwand. Denken Sie daran, dass das Ergebnis dieser Berechnung nicht als echte Wahrscheinlichkeit genutzt wird. Stattdessen werden die Wahrscheinlichkeiten für jede Kategorie getrennt ermittelt und dann die Ergebnisse miteinander verglichen. Da *Pr(Dokument)* immer gleich ist – egal, für welche Kategorie es berechnet wird –, wird es die Ergebnisse immer auf die gleiche Weise beeinflussen, und Sie können es ignorieren.

Die Methode `prob` berechnet die Wahrscheinlichkeit für die Kategorie und liefert das Produkt aus *Pr(Dokument | Kategorie)* und *Pr(Kategorie)* zurück. Fügen Sie diese Methode der Klasse `naivebayes` hinzu:

```
def prob(self,item,cat):
  catprob=self.catcount(cat)/self.totalcount( )
  docprob=self.docprob(item,cat)
  return docprob*catprob
```

Probieren Sie diese Funktion in Python aus, um zu sehen, wie sich die Werte für verschiedene Strings und Kategorien unterscheiden:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.pyc'>
>>> cl=docclass.naivebayes(docclass.getwords)
>>> docclass.sampletrain(cl)
```

e-bol.net

```
>>> cl.prob('quick rabbit','good')
0.15624999999999997
>>> cl.prob('quick rabbit','bad')
0.050000000000000003
```

Basierend auf den Trainingsdaten, sieht es so aus, als wäre die Phrase »quick rabbit« ein deutlich besserer Kandidat für die gute Kategorie als für die schlechte.

Auswählen einer Kategorie

Der letzte Schritt beim Aufbau des naiven Bayes-Klassifizierers ist die eigentliche Entscheidung, in welche Kategorie ein neues Element gehört. Der einfachste Ansatz würde die Wahrscheinlichkeit dieses Elements für jede der Kategorien berechnen und diejenige auswählen, bei der sie am höchsten ist. Wenn Sie nur versuchen, den besten Platz zu finden, um dort ein Element unterzubringen, wäre das ausreichend, aber in vielen Anwendungen sind die Kategorien nicht gleichwertig, und in manchen Anwendungen ist es für den Klassifizierer besser zuzugeben, dass er die Antwort nicht weiß, anstatt zu entscheiden, dass die Antwort eine Kategorie ist, die nur eine minimal höhere Wahrscheinlichkeit hat.

Beim Filtern von Spam ist es viel wichtiger zu vermeiden, gute E-Mails als Spam zu klassifizieren, als jede einzelne Spam-Nachricht zu erwischen. Die gelegentlich in Ihrer Inbox auftauchenden Spam-Mails sind tolerierbar, aber eine wichtige E-Mail, die automatisch als Spam ausgefiltert wurde, würde vollständig übersehen werden. Wenn Sie Ihren Junk-Ordner erst nach wichtigen E-Mails absuchen müssen, lohnt es sich gar nicht erst, einen Spam-Filter einzurichten.

Um dieses Problem zu lösen, können Sie für jede Kategorie einen Minimalwert festlegen. Soll ein neues Element in eine bestimmte Kategorie eingestuft werden, muss seine Wahrscheinlichkeit um einen gewissen Wert höher sein als der für die anderen Kategorien. Dieser Wert ist der *Schwellenwert* (*Threshold*). Beim Filtern von Spam könnte der Schwellenwert für eine Einstufung in die schlechte Kategorie 3 sein, sodass die Wahrscheinlichkeit für »schlecht« dreimal höher sein müsste als die für »gut«. Der Schwellenwert für »gut« könnte auf 1 gesetzt werden, sodass alles als gut definiert würde, dessen Wahrscheinlichkeit dafür überhaupt höher ist als für die schlechte Kategorie. Jede Nachricht, bei der die Wahrscheinlichkeit für »schlecht« höher ist, aber nicht dreimal höher, würde als unbekannt klassifiziert werden.

Um diese Schwellenwerte festzulegen, fügen Sie `classifier` eine neue Instanzvariable hinzu, indem Sie die Initialisierungsmethode anpassen:

```
def __init__(self,getfeatures):
  classifier.__init__(self,getfeatures)
  self.thresholds={}
```

e-bol.net

Dazu gibt es noch ein paar einfache Methoden, um die Werte zu setzen und auszulesen, wobei 1.0 als Standard zurückgegeben wird:

```
def setthreshold(self,cat,t):
  self.thresholds[cat]=t

def getthreshold(self,cat):
  if cat not in self.thresholds: return 1.0
  return self.thresholds[cat]
```

Jetzt können Sie die Methode `classify` schreiben. Sie wird die Wahrscheinlichkeit für jede Kategorie berechnen und bestimmen, welche die größte ist und ob sie um mehr als den Schwellenwert größer ist als die nächste Kategorie. Wenn keine der Kategorien dies erfüllt, liefert die Methode nur die Standardwerte zurück. Fügen Sie diesen Code `classifier` hinzu:

```
def classify(self,item,default=None):
  probs={}
  # Kategorie mit der höchsten Wahrscheinlichkeit finden.
  max=0.0
  for cat in self.categories( ):
    probs[cat]=self.prob(item,cat)
    if probs[cat]>max:
      max=probs[cat]
      best=cat

  # Sicherstellen, dass die Wahrscheinlichkeit größer ist als
  # threshold * nächstbeste Wahrscheinlichkeit.
  for cat in probs:
    if cat==best: continue
    if probs[cat]*self.getthreshold(best)>probs[best]: return default
  return best
```

Und fertig! Sie haben nun ein vollständiges System zum Klassifizieren von Dokumenten aufgebaut. Man kann dies noch erweitern, um andere Dinge zu klassifizieren, indem man unterschiedliche Methoden zum Bestimmen der Merkmale schreibt. Testen Sie den Klassifizierer in Ihrer Python-Session:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.pyc'>
>>> cl=docclass.naivebayes(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> cl.classify('quick rabbit',default='unknown')
'good'
>>> cl.classify('quick money',default='unknown')
'bad'
>>> cl.setthreshold('bad',3.0)
>>> cl.classify('quick money',default='unknown')
'unknown'
>>> for i in range(10): docclass.sampletrain(cl)
...
>>> cl.classify('quick money',default='unknown')
'bad'
```

e-bol.net

Natürlich können Sie die Schwellenwerte anpassen und probieren, wie dies die Ergebnisse beeinflusst. Manche Spam-Filter-Plugins ermöglichen dem Benutzer die Kontrolle über die Schwellenwerte, sodass man sie anpassen kann, wenn sie zu viel Spam durchlassen oder gute Nachrichten als Spam kategorisieren. Die Schwellenwerte werden sich auch für andere Anwendungen unterscheiden, die Dokumente filtern – manchmal werden alle Kategorien gleich sein oder das Einstufen in »unknown« wird nicht akzeptabel sein.

Die Fisher-Methode

Die *Fisher-Methode*, benannt nach R. A. Fisher, ist eine alternative Methode, die sehr genaue Ergebnisse zurückliefert, insbesondere beim Spam-Filtern. Diese Methode wird auch von *SpamBayes* genutzt, einem Outlook-Plugin, das in Python geschrieben ist. Anders als der naive Bayes-Filter, der die Merkmalswahrscheinlichkeiten nutzt, um die Wahrscheinlichkeit für das Gesamtdokument zu erstellen, berechnet die Fisher-Methode die Wahrscheinlichkeit einer Kategorie für jedes Merkmal im Dokument, kombiniert dann die Wahrscheinlichkeiten und prüft, ob das Set mit Wahrscheinlichkeiten einem zufälligen Set ähnelt oder nicht. Auch wenn es sich dabei um eine komplexere Methode handelt, lohnt es sich, sie kennenzulernen, da sie eine größere Flexibilität beim Wählen der Grenzwerte für die Kategorisierung ermöglicht.

Kategorie-Wahrscheinlichkeiten für Merkmale

Beim weiter oben besprochenen naiven Bayes-Filter haben Sie alle Ergebnisse von *Pr(Merkmal | Kategorie)* kombiniert, um eine Gesamtwahrscheinlichkeit für das Dokument zu erhalten, und sie dann am Ende vertauscht. In diesem Abschnitt werden Sie zunächst berechnen, wie wahrscheinlich es ist, dass ein Dokument in eine Kategorie passt, wenn sich ein bestimmtes Merkmal in dieser Kategorie befindet – also *Pr(Kategorie | Merkmal)*. Kommt das Wort »casino« in 500 Dokumenten vor und befinden sich 499 davon in der schlechten Kategorie, wird »casino« für »schlecht« einen Wert erhalten, der sehr nahe an 1 liegt.

Normalerweise würde man *Pr(Kategorie | Merkmal)* so berechnen:

> *(Anzahl der Dokumente in dieser Kategorie mit diesem Merkmal) /*
> *(Gesamtanzahl an Dokumenten mit diesem Merkmal)*

Diese Berechnung berücksichtigt aber nicht die Wahrscheinlichkeit, dass Sie viel mehr Dokumente in einer Kategorie haben als in einer anderen. Wenn Sie viele gute Dokumente und nur ein paar schlechte haben, wird ein Wort, das in all Ihren schlechten Dokumenten auftaucht, ziemlich sicher eine hohe Wahrscheinlichkeit für »schlecht« haben, selbst wenn die Nachricht selbst wohl gut ist. Die Methoden funktionieren besser, wenn sie davon ausgehen, dass Sie zukünftig eine gleich große

e-bol.net

Anzahl an Dokumenten in jeder Kategorie besitzen werden, da sie dadurch einen Vorteil aus den Merkmalen ziehen können, die die Kategorien unterscheiden.

Um diese Normalisierung durchzuführen, berechnet die Methode drei Dinge:

- *clf = Pr(Merkmal | Kategorie) für diese Kategorie*
- *freqsum = Summe von Pr(Merkmal | Kategorie) für alle Kategorien*
- *cprob = clf / (clf+nclf)*

Erstellen Sie eine neue Unterklasse von `classifier` namens `fisherclassifier` in *docclass.py* und fügen Sie diese Methode hinzu:

```
class fisherclassifier(classifier):
  def cprob(self,f,cat):
    # Häufigkeit dieses Merkmals in dieser Kategorie.
    clf=self.fprob(f,cat)
    if clf==0: return 0

    # Häufigkeit dieses Merkmals in allen Kategorien.
    freqsum=sum([self.fprob(f,c) for c in self.categories( )])

    # Die Wahrscheinlichkeit ist die Häufigkeit in dieser Kategorie,
    # geteilt durch die Gesamthäufigkeit.
    p=clf/(freqsum)

    return p
```

Diese Funktion wird die Wahrscheinlichkeit zurückliefern, dass ein Element mit dem entsprechenden Merkmal zu der angegebenen Kategorie gehört, wenn man davon ausgeht, dass es eine gleich große Anzahl von Elementen in jeder Kategorie gibt. Sie können sich diese Werte in Ihrer Python-Session ausgeben lassen:

```
>>> reload(docclass)
>>> cl=docclass.fisherclassifier(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> cl.cprob('quick','good')
0.57142857142857151
>>> cl.cprob('money','bad')
1.0
```

Diese Methode zeigt uns, dass Dokumente mit dem Wort »casino« eine Wahrscheinlichkeit von 0,9 haben, dass es sich um Spam handelt. Das passt zu den Trainingsdaten, aber auch hier hadern wir mit dem Problem, dass die Wörter nur sehr selten vorkamen und die Wahrscheinlichkeiten zu sehr überbewertet werden. Daher ist es wie beim letzten Mal besser, die gewichteten Wahrscheinlichkeiten zu nutzen, bei denen alle bei 0,5 beginnen und mit zunehmendem Training die »echten« Wahrscheinlichkeiten erreicht werden.

```
>>> cl.weightedprob('money','bad',cl.cprob)
0.75
```

e-bol.net

Kombinieren der Wahrscheinlichkeiten

Sie müssen nun die Wahrscheinlichkeiten der einzelnen Merkmale kombinieren, um eine Gesamtwahrscheinlichkeit zu erhalten. Theoretisch können Sie alle miteinander multiplizieren, wodurch Sie eine Wahrscheinlichkeit erhalten, die Sie nutzen können, um diese Kategorie mit anderen Kategorien zu vergleichen. Aber da die Merkmale nicht unabhängig voneinander sind, wäre dies keine echte Wahrscheinlichkeit. Es würde allerdings so wie der Bayes-Klassifizierer funktionieren, den Sie im vorigen Abschnitt gebaut haben. Der von der Fisher-Methode zurückgegebene Wert ist eine deutlich bessere Schätzung der Wahrscheinlichkeit, die sehr nützlich sein kann, wenn man Ergebnisse ausgeben möchte oder sich für einen Schwellenwert entscheiden muss.

Die Fisher-Methode multipliziert alle Wahrscheinlichkeiten miteinander, nimmt dann den natürlichen Logarithmus (`math.log` in Python) und multipliziert das Ergebnis mit –2. Fügen Sie diese Methode `fisherclassifier` hinzu, um die Berechnungen zu ermöglichen:

```
def fisherprob(self,item,cat):
  # Alle Wahrscheinlichkeiten miteinander multiplizieren.
  p=1
  features=self.getfeatures(item)
  for f in features:
    p*=(self.weightedprob(f,cat,self.cprob))

  # Den natürlichen Logarithmus bestimmen und mit -2 multiplizieren.
  fscore=-2*math.log(p)

  # Mit der inversen chi2-Funktion eine Wahrscheinlichkeit bestimmen.
  return self.invchi2(fscore,len(features)*2)
```

Fisher hat gezeigt, dass das Ergebnis dieser Berechnung bei unabhängigen und zufälligen Wahrscheinlichkeiten zu einer *Chi-Quadrat-Verteilung* passt. Sie erwarten vielleicht, dass ein Element, das nicht zu einer bestimmten Kategorie passt, Wörter mit verschiedenen Merkmalswahrscheinlichkeiten für diese Kategorie enthält (was mehr oder weniger zufällig erscheint) und dass ein Element, das zu dieser Kategorie gehört, viele Merkmale mit hohen Wahrscheinlichkeiten hat. Indem das Ergebnis der Fisher-Berechnung der *inversen Chi-Quadrat-Funktion* übergeben wird, erhalten Sie die Wahrscheinlichkeit, dass ein zufälliges Set von Wahrscheinlichkeiten solch einen hohen Wert liefern würde.

Fügen Sie die inverse Chi-Quadrat-Funktion der Klasse `fisherclassifier` hinzu:

```
def invchi2(self,chi,df):
  m = chi / 2.0
  sum = term = math.exp(-m)
  for i in range(1, df//2):
      term *= m / i
      sum += term
  return min(sum, 1.0)
```

e-bol.net

Auch hier können Sie diese Funktion in Ihrer Python-Session ausprobieren und testen, wie die Fisher-Methode ein paar Beispiel-Strings bewertet:

```
>>> reload(docclass)
>>> cl=docclass.fisherclassifier(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> cl.cprob('quick','good')
0.57142857142857151
>>> cl.fisherprob('quick rabbit','good')
0.78013986588957995
>>> cl.fisherprob('quick rabbit','bad')
0.35633596283335256
```

Wie Sie sehen können, sind diese Ergebnisse immer zwischen 0 und 1. Sie sind für sich allein ein guter Maßstab dafür, wie gut ein Dokument in eine Kategorie passt. Daher kann die Klassifikation selbst noch ausgefeilter werden.

Elemente klassifizieren

Sie können die Werte, die von `fisherprob` zurückgegeben werden, nutzen, um die Klassifikation zu bestimmen. Anstatt Multiplikationsschwellenwerte wie beim Bayes-Filter zu nutzen, können Sie die unteren Grenzen für jede Klassifikation festlegen. Der Klassifizierer wird den höchsten Wert innerhalb seiner Grenzen zurückgeben. Im Spam-Filter setzen Sie vielleicht den unteren Wert für die schlechte Klassifizierung ziemlich hoch, etwa auf 0,6. Das Minimum für die gute Klassifizierung können Sie niedriger setzen, zum Beispiel auf 0,2. Damit verringern Sie die Gefahr, dass gute E-Mails als schlecht klassifiziert werden, lassen aber ein paar Spam-E-Mails in der Inbox zu. Alles, was unter 0,2 für gut und unter 0,6 für schlecht bewertet wird, würde dann als unbekannt klassifiziert.

Erstellen Sie eine Methode `init` in `fisherclassifier` mit einer weiteren Variablen, um die Schwellenwerte abzulegen:

```
def __init__(self,getfeatures):
  classifier.__init__(self,getfeatures)
  self.minimums={}
```

Fügen Sie noch ein paar Methoden hinzu, um diese Werte zu setzen und auszulesen, wobei ein Standardwert von 0 genutzt wird:

```
def setminimum(self,cat,min):
  self.minimums[cat]=min

def getminimum(self,cat):
  if cat not in self.minimums: return 0
  return self.minimums[cat]
```

Schließlich verwenden Sie eine weitere Methode, um die Wahrscheinlichkeiten für jede Kategorie zu berechnen und das beste Ergebnis zu ermitteln, das das angegebene Minimum überschreitet:

e-bol.net

```
def classify(self,item,default=None):
  # Schleife durchlaufen, um das beste Ergebnis zu erhalten.
  best=default
  max=0.0
  for c in self.categories( ):
    p=self.fisherprob(item,c)
    # Sicherstellen, dass das Minimum überschritten wird.
    if p>self.getminimum(c) and p>max:
      best=c
      max=p
  return best
```

Nun können Sie den Klassifizierer mit den Testdaten und der Fisher-Bewertungsmethode ausprobieren. Geben Sie das Folgende in Ihrer Python-Session ein:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.py'>
>>> docclass.sampletrain(cl)
>>> cl.classify('quick rabbit')
'good'
>>> cl.classify('quick money')
'bad'
>>> cl.setminimum('bad',0.8)
>>> cl.classify('quick money')
'good'
>>> cl.setminimum('good',0.4)
>>> cl.classify('quick money')
>>>
```

Die Ergebnisse entsprechen denen des naiven Bayes-Klassifizierers. Der Fisher-Klassifizierer ist wohl beim Spam-Filtern in der Praxis schneller, allerdings wird dies bei solch einem kleinen Trainingsdatenbestand nicht weiter auffallen. Es hängt von Ihrer Anwendung ab, welchen Klassifizierer Sie nutzen sollten, und es lässt sich im Voraus nur schlecht sagen, welcher schneller sein wird und welche Schwellenwerte passen. Zum Glück wird Ihnen der hier vorgestellte Code es sehr einfach machen, mit den beiden Algorithmen und verschiedenen Einstellungen zu experimentieren.

Die trainierten Klassifizierer persistieren

In einer echten Anwendung ist es recht unwahrscheinlich, dass das gesamte Training und die Klassifikation in einer einzelnen Session durchgeführt wird. Wenn der Klassifizierer als Teil einer webbasierten Anwendung genutzt wird, wird er sehr wahrscheinlich sein trainiertes Wissen abspeichern müssen, das der Benutzer ihm hat angedeien lassen, und die Daten wieder laden, wenn sich der Benutzer das nächste Mal anmeldet.

e-bol.net

SQLite nutzen

Dieser Abschnitt wird Ihnen zeigen, wie Sie die Trainingsinformationen für Ihren Klassifizierer mithilfe einer Datenbank persistieren können – in diesem Fall mit SQLite. Wenn Ihre Anwendung viele Benutzer enthält, die den Klassifizierer gleichzeitig trainieren und benutzen wollen, ist es äußerst sinnvoll, die Summen in einer Datenbank abzuspeichern. SQLite ist die gleiche Datenbank, die wir schon in Kapitel 4 verwendet haben. Sie müssen *pysqlite* herunterladen und installieren, wenn Sie es noch nicht getan haben – Informationen dazu finden Sie in Anhang A. Der Zugriff auf SQLite von Python aus entspricht dem auf andere Datenbanken, daher sollte es leicht von der Hand gehen.

Um *pysqlite* zu importieren, fügen Sie diese Anweisung am Anfang von *docclass.py* hinzu:

```
from pysqlite2 import dbapi2 as sqlite
```

Der Code in diesem Abschnitt wird die Dictionary-Strukturen, die bisher in der Klasse `classifier` genutzt wurden, durch einen persistenten Datenspeicher ersetzen. Fügen Sie eine Methode `classifier` hinzu, die eine Datenbank für diesen Klassifizierer öffnet und die Tabellen anlegt, falls sie noch nicht vorhanden sind. Die Tabellen entsprechen den Strukturen der Dictionaries, die sie ersetzen werden:

```
def setdb(self,dbfile):
  self.con=sqlite.connect(dbfile)
  self.con.execute('create table if not exists fc(feature,category,count)')
  self.con.execute('create table if not exists cc(category,count)')
```

Wenn Sie diesen Code mit einer anderen Datenbank einsetzen wollen, müssen Sie die Anweisungen `create table` anpassen, damit sie auch bei Ihnen funktionieren.

Verändern Sie alle Hilfsmethoden zum Auslesen und Erhöhen der Zähler:

```
def incf(self,f,cat):
  count=self.fcount(f,cat)
  if count==0:
    self.con.execute("insert into fc values ('%s','%s',1)"
                     % (f,cat))
  else:
    self.con.execute(
      "update fc set count=%d where feature='%s' and category='%s'"
      % (count+1,f,cat))

def fcount(self,f,cat):
  res=self.con.execute(
    'select count from fc where feature="%s" and category="%s"'
    %(f,cat)).fetchone( )
  if res==None: return 0
  else: return float(res[0])

def incc(self,cat):
```

e-bol.net

```
    count=self.catcount(cat)
    if count==0:
      self.con.execute("insert into cc values ('%s',1)" % (cat))
    else:
      self.con.execute("update cc set count=%d where category='%s'"
                       % (count+1,cat))

  def catcount(self,cat):
    res=self.con.execute('select count from cc where category="%s"'
                         %(cat)).fetchone()
    if res==None: return 0
    else: return float(res[0])
```

Die Methoden, die die Listen aller Kategorien und die Gesamtanzahl an Dokumenten zurückgeben, müssen auch ausgetauscht werden:

```
  def categories(self):
    cur=self.con.execute('select category from cc');
    return [d[0] for d in cur]

  def totalcount(self):
    res=self.con.execute('select sum(count) from cc').fetchone();
    if res==None: return 0
    return res[0]
```

Schließlich müssen Sie nach dem Training noch ein `commit` hinzufügen, sodass die Daten auch gespeichert werden, nachdem die Zähler angepasst wurden. Fügen Sie diese Zeile am Ende der Methode `train` in `classifier` hinzu:

```
    self.con.commit()
```

Das ist alles! Nachdem Sie einen Klassifizierer initialisiert haben, müssen Sie die Methode `setdb` mit dem Namen einer Datenbankdatei aufrufen. Die Trainingsdaten werden automatisch gespeichert und können von anderen Personen genutzt werden. Sie können auch die Trainingsdaten von einem Klassifizierer nutzen, um Klassifikationen mit einem anderen Klassifizierer durchzuführen:

```
>>> reload(docclass)
<module 'docclass' from 'docclass.py'>
>>> cl=docclass.fisherclassifier(docclass.getwords)
>>> cl.setdb('test1.db')
>>> docclass.sampletrain(cl)
>>> cl2=docclass.naivebayes(docclass.getwords)
>>> cl2.setdb('test1.db')
>>> cl2.classify('quick money')
u'bad'
```

Blog-Feeds filtern

Um den Klassifizierer mit echten Daten auszutesten und die verschiedenen Möglichkeiten zu zeigen, bei denen er sich nutzen lässt, können Sie ihn auf die Einträge eines Blog- oder eines anderen RSS-Feeds ansetzen. Dazu benötigen Sie den Univer-

e-bol.net

sal Feed Parser, den wir in Kapitel 3 verwendet haben. Wenn Sie ihn noch nicht heruntergeladen haben, holen Sie das nach unter *http://feedparser.org*. Mehr Informationen zur Installation des Feed Parser finden Sie in Anhang A.

Auch wenn ein Blog in seinen Einträgen nicht notwendigerweise Spam enthält, gibt es doch in vielen Blogs ein paar Artikel, die Sie interessieren, und ein paar, die Sie nicht interessant finden. Das kann daran liegen, dass Sie nur Artikel einer bestimmten Kategorie oder von einem bestimmten Autor lesen wollen, aber häufig ist es auch komplizierter. Auch hier können Sie bestimmte Regeln dazu angeben, was Sie interessiert und was nicht – vielleicht lesen Sie ein Gadget-Blog und haben kein Interesse an Einträgen, die die Wörter »cell phone« enthalten – aber es ist viel weniger aufwendig, den von Ihnen erstellten Klassifizierer zu nutzen, der die Regeln für Sie herausfindet.

Ein Vorteil beim Klassifizieren von Einträgen in einem RSS-Feed ist folgender: Wenn Sie ein Blog-Suchtool wie Google Blog Search nutzen, können Sie die Ergebnisse Ihrer Suchen in einen Feed-Reader packen. Viele tun das, um Produkte, für sie interessante Dinge oder sogar ihren eigenen Namen zu verfolgen. Sie werden allerdings feststellen, dass Spam-basierte oder nutzlose Blogs versuchen, Geld mit dem Blog-Traffic zu machen, und daher in diesen Suchen ebenfalls auftauchen.

Für dieses Beispiel können Sie jeden beliebigen Feed nutzen, auch wenn viele Feeds zu wenige Einträge für ein effektives Training enthalten. Dieses spezielle Beispiel nutzt die Ergebnisse einer Suche per Google Blog Search nach dem Wort »Python« im RSS-Format. Sie können diese Ergebnisse unter *http://kiwitobes.com/feeds/python_search.xml* herunterladen.

Erstellen Sie eine neue Datei namens *feedfilter.py* und fügen Sie den folgenden Code hinzu:

```
import feedparser
import re

# Erwartet einen Dateinamen oder die URL eines Blog-Feeds und
# klassifiziert die Einträge.
def read(feed,classifier):
  # Feed-Einträge holen und sie durchlaufen.
  f=feedparser.parse(feed)
  for entry in f['entries']:
    print
    print '-----'
    # Inhalt des Eintrags ausgeben.
    print 'Titel:       '+entry['title'].encode('utf-8')
    print 'Herausgeber: '+entry['publisher'].encode('utf-8')
    print
    print entry['summary'].encode('utf-8')


    # Texte kombinieren, um ein Element für den Klassifizierer zu erstellen.
    fulltext='%s\n%s\n%s' % (entry['title'],entry['publisher'],entry['summary'])
```

e-bol.net

```
    # Beste Einschätzung für die Kategorie abgeben.
    print 'Einschätzung: '+str(classifier.classify(fulltext))

    # Benutzer nach der richtigen Kategorie fragen und damit trainieren.
    cl=raw_input('Kategorie eingeben: ')
    classifier.train(fulltext,cl)
```

Diese Funktion durchläuft alle Einträge und nutzt den Klassifizierer, um die beste Einschätzung der Klassifikation zu erhalten. Dann zeigt sie diese an und bittet den Benutzer, die korrekte Kategorie anzugeben. Wenn Sie das für einen neuen Klassifizierer durchziehen, werden die Einschätzungen zunächst zufällig sein, sich mit der Zeit aber verbessern.

Der Klassifizierer, den Sie gebaut haben, ist komplett generisch. Auch wenn das Filtern von Spam als Beispiel genutzt wurde, um zu erläutern, was jeder Codeabschnitt für Aufgaben hat, können die Kategorien beliebig sein. Nutzen Sie *python_search.xml*, haben Sie vielleicht vier Kategorien – eine für die Programmiersprache, eine für Monty Python, eine für Python-Schlangen und eine für alles andere. Probieren Sie einmal, den interaktiven Filter in Ihrer Python-Session auszuführen, indem Sie einen Klassifizierer erstellen und ihn an `feedfilter` weitergeben:

```
>>> import feedfilter
>>> cl=docclass.fisherclassifier(docclass.getwords)
>>> cl.setdb('python_feed.db')  # nur wenn Sie SQLite nutzen
>>> feedfilter.read('python_search.xml',cl)

-----
Titel:       My new baby boy!
Herausgeber: Shetan Noir, the zombie belly dancer! - MySpace Blog

This is my new baby, Anthem. He is a 3 and half month old ball <b>python</b>,
orange shaded normal pattern. I have held him about 5 times since I brought him
home tonight at 8:00pm...
Einschätzung: None
Kategorie eingeben: Schlange

-----
Titel:       If you need a laugh...
Herausgeber: Kate&#39;s space

Even does 'funny walks' from Monty <b>Python</b>. He talks about all the ol'
Einschätzung: Schlange
Kategorie eingeben: Monty

-----
Titel:       And another one checked off the list..New pix comment ppl
Herausgeber: And Python Guru - MySpace Blog

Now the one of a kind NERD bred Carplot male is in our possesion. His name is
Broken (not because he is sterile) lol But check out the pic and leave one
Einschätzung: Schlange
Kategorie eingeben: Schlange
```

e-bol.net

Sie werden sehen, dass sich die Einschätzungen mit der Zeit verbessern. Es gibt nicht viele Beispiele für Schlangen, daher irrt sich der Klassifizierer hier gern, insbesondere da die Kategorie sich auch noch in Schlangen als Haustiere und in modebezogene Posts aufteilt. Nachdem Sie das Training durchlaufen haben, können Sie die Wahrscheinlichkeiten für ein bestimmtes Merkmal erhalten – sowohl für ein Wort bei gegebener Kategorie als auch die Wahrscheinlichkeit für eine Kategorie bei einem gegebenen Wort:

```
>>> cl.cprob('python','prog')
0.33333333333333331
>>> cl.cprob('python','snake')
0.33333333333333331
>>> cl.cprob('python','monty')
0.33333333333333331
>>> cl.cprob('eric','monty')
1.0
>>> cl.fprob('eric','monty')
0.25
```

Die Wahrscheinlichkeiten für das Wort »python« sind gleich verteilt, da jeder Eintrag dieses Wort enthält. Das Wort »Eric« taucht in 25% der Einträge auf, die zu Monty Python gehören, aber in keinem anderen Eintrag. Daher ist die Wahrscheinlichkeit für das Wort in dieser Kategoire 0,25, während die Kategorie für das Wort 1,0 ist.

Merkmalserkennung verbessern

In allen bisherigen Beispielen nutzt die Funktion zum Erstellen einer Liste der Merkmale eine einfache Teilung bei Zeichen, die keine Buchstaben sind, um die Wörter aufzuteilen. Die Funktion wandelt alle Wörter in Kleinbuchstaben um, daher gibt es keine Möglichkeit, die exzessive Verwendung von Großbuchstaben zu erkennen. Es gibt viele Möglichkeiten, dies zu verbessern:

- Ohne Token in Groß- und Kleinbuchstaben tatsächlich als unterschiedliche Merkmale anzusehen, können Sie die Häufigkeit vieler Wörter in Großbuchstaben trotzdem als Merkmal verwenden.
- Nutzen Sie neben den einzelnen Wörtern Kombinationen von Wörtern.
- Nehmen Sie Rücksicht auf weitere Metainformationen, zum Beispiel wer eine E-Mail verschickt hat oder in welcher Kategorie ein Blog gepostet wurde, und fügen Sie dies als Merkmale hinzu.
- Lassen Sie URLs und Zahlen intakt.

Denken Sie daran, dass es nicht nur einfach darum geht, die Merkmale spezieller zu machen. Sie müssen stattdessen in mehreren Dokumenten auftauchen, damit sie für den Klassifizierer von Wert sind.

e-bol.net

Die Klasse `classifier` erwartet eine Funktion als `getfeatures` und führt sie für das übergebene Element aus, wodurch eine Liste oder ein Dictionary mit allen Merkmalen für dieses Element entstehen soll. Da das Vorgehen so generisch ist, können Sie leicht eine Funktion erstellen, die mit Elementen umgeht, die komplizierter sind als einfache Strings. Wenn Sie zum Beispiel Einträge in einem Blog-Feed klassifizieren, können Sie eine Funktion nutzen, die den gesamten Eintrag statt des extrahierten Texts nutzt und markiert, wo die verschiedenen Wörter herkommen. Sie können auch Wortpaare aus dem Text herausziehen und nur einzelne Wörter aus dem Betreff. Allerdings ist es vermutlich ziemlich unnötig, das Feld `creator` in Tokens umzuwandeln, da die Postings eines »John Smith« Ihnen vermutlich nicht viel darüber sagen werden, wie jemand anderes postet, dessen Vorname John ist.

Fügen Sie diese neue Funktion zum Extrahieren von Merkmalen der Datei *feedfilter.py* hinzu. Beachten Sie, dass sie keinen String, sondern einen Feed-Eintrag als Parameter erwartet:

```
def entryfeatures(entry):
  splitter=re.compile('\\W*')
  f={}

  # Wörter aus dem Titel extrahieren und kenntlich machen.
  titlewords=[s.lower( ) for s in splitter.split(entry['title'])
          if len(s)>2 and len(s)<20]
  for w in titlewords: f['Title:'+w]=1

  # Wörter der Zusammenfassung extrahieren.
  summarywords=[s.lower( ) for s in splitter.split(entry['summary'])
          if len(s)>2 and len(s)<20]

  # Wörter in Großbuchstaben zählen.
  uc=0
  for i in range(len(summarywords)):
    w=summarywords[i]
    f[w]=1
    if w.isupper( ): uc+=1

    # Wortpaare in der Zusammenfassung als Merkmale extrahieren.
    if i<len(summarywords)-1:
      twowords=' '.join(summarywords[i:i+1])
      f[twowords]=1

  # Autor und Herausgeber als Ganzes nutzen.
  f['Publisher:'+entry['publisher']]=1

  # UPPERCASE ist ein virtuelles Wort, das angibt, ob zu viel geschrien wird.
  if float(uc)/len(summarywords)>0.3: f['UPPERCASE']=1

  return f
```

e-bol.net

Diese Funktion extrahiert die Wörter aus dem Titel und der Zusammenfassung, so wie es `getwords` bisher schon tat. Dann markiert sie alle Wörter im Titel als solche und fügt sie als Merkmale hinzu. Die Wörter in der Zusammenfassung werden als Merkmale eingetragen, und Paare zusammenhängender Wörter werden ebenfalls hinzugefügt. Die Funktion nutzt noch den Ersteller und den Herausgeber als Merkmale, ohne sie aufzuteilen, und schließlich zählt sie die Anzahl der Wörter in der Zusammenfassung, die in Großbuchstaben geschrieben sind. Wenn mehr als 30% der Wörter nur aus Großbuchstaben bestehen, fügt die Funktion ein zusätzliches Merkmal namens UPPERCASE hinzu. Anders als eine Regel, die besagt, dass Wörter in Großbuchstaben etwas Bestimmtes bedeuten, ist dies hier nur ein zusätzliches Merkmal, das der Klassifizierer zum Trainieren nutzen kann – in manchen Fällen kann er entscheiden, dass es nicht nützlich ist, um die Dokumentenkategorien zu unterscheiden.

Wenn Sie diese neue Version zusammen mit `filterfeed` nutzen wollen, müssen Sie die Funktion anpassen, die die Einträge an den Klassifizierer übergibt. Ändern Sie das Ende so ab:

```
# Beste Einschätzung für die Kategorie abgeben.
print 'Einschätzung: '+str(classifier.classify(entry))

# Benutzer nach der richtigen Kategorie fragen und damit trainieren.
cl=raw_input('Kategorie eingeben: ')
classifier.train(entry,cl)
```

Sie können dann den Klassifizierer so initialisieren, dass er `entryfeatures` als Methode zum Extrahieren von Merkmalen nutzt:

```
>>> reload(feedfilter)
<module 'feedfilter' from 'feedfilter.py'>
>>> cl=docclass.fisherclassifier(feedfilter.entryfeatures)
>>> cl.setdb('python_feed.db')  # nur wenn Sie die DB-Version nutzen
>>> feedfilter.read('python_search.xml',cl)
```

Sie können noch viel mehr mit Merkmalen anstellen. Das grundlegende Framework, das Sie schon aufgebaut haben, ermöglicht Ihnen, eine Funktion zum Extrahieren der Merkmale zu definieren und den Klassifizierer so einzurichten, dass er diese Funktion nutzt. Er wird dann jedes Objekt klassifizieren, das Sie ihm übergeben, solange die von Ihnen angegebene Funktion zum Extrahieren der Merkmale eine Liste von Merkmalen für das Objekt zurückgeben kann.

Akismet

Akismet ist ein kleiner Abstecher bei den Untersuchungen zu Text-Klassifizierungsalgorithmen, aber für eine bestimmte Art von Anwendungen kann es Ihre Anforderungen zum Filtern von Spam mit minimalem Aufwand lösen und die Notwendigkeit beseitigen, sich Ihren eigenen Klassifizierer bauen zu müssen.

e-bol.net

Akismet begann als Plugin für WordPress, das den Leuten erlaubte, Spam-Kommentare zu melden, die in ihren Blogs gepostet wurden, und neue Kommentare in Abhängigkeit von ihrer Ähnlichkeit zu von anderen Personen gemeldetem Spam zu filtern. Nun ist die API offen, und Sie können Akismet mit beliebigen Strings befragen, um herauszufinden, ob es denkt, dass dieser String Spam sei.

Das Erste, was Sie brauchen, ist ein Akismet-API-Schlüssel, den Sie unter *http://akismet.com* erhalten können. Diese Schlüssel sind für den persönlichen Bedarf kostenlos, und es gibt für die kommerzielle Nutzung diverse Varianten. Die Akismet-API wird über normale HTTP-Requests angesprochen, wobei es für die verschiedensten Sprachen Bibliotheken gibt. Die in diesem Abschnitt genutzte ist verfügbar unter *http://kemayo.wordpress.com/2005/12/02/akismet-py*. Laden Sie *akismet.py* herunter und kopieren Sie die Datei in Ihr Codeverzeichnis oder in das Verzeichnis mit den Python-Bibliotheken.

Die Nutzung der API ist sehr einfach. Erstellen Sie eine neue Datei namens *akismettest.py* und fügen Sie diese Funktion hinzu:

```
import akismet

defaultkey = "YOURKEYHERE"
pageurl="http://yoururlhere.com"

defaultagent="Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) "
defaultagent+="Gecko/20060909 Firefox/1.5.0.7"

def isspam(comment,author,ipaddress,
           agent=defaultagent,
           apikey=defaultkey):
  try:
    valid = akismet.verify_key(apikey,pageurl)
    if valid:
      return akismet.comment_check(apikey,pageurl,
        ipaddress,agent,comment_content=comment,
        comment_author_email=author,comment_type="comment")
    else:
      print 'Ungültiger Schlüssel'
      return False
  except akismet.AkismetError, e:
      print e.response, e.statuscode
      return False
```

Sie haben jetzt eine Methode, die Sie mit einem beliebigen String aufrufen können, um herauszufinden, ob er Texten in Blog-Kommentaren ähnelt. Probieren Sie es in Ihrer Python-Session aus:

```
>>> import akismettest
>>> msg='Make money fast! Online Casino!'
>>> akismettest.isspam(msg,'spammer@spam.com','127.0.0.1')
True
```

e-bol.net

Experimentieren Sie mit verschiedenen Benutzernamen, Agents und IP-Adressen, um herauszufinden, wie sich die Ergebnisse unterscheiden.

Da Akismet hauptsächlich für Spam-Kommentare in Blogs genutzt wird, funktioniert es für andere Arten von Dokumenten, wie zum Beispiel E-Mails, nicht so gut. Und anders als beim Klassifizieren können Sie weder an den Parametern drehen noch die Berechnungen einsehen, aufgrund deren es die Antworten liefert. Aber für das Filtern von Spam-Kommentaren ist es ausgesprochen exakt, und es lohnt einen Test mit Ihrer Anwendung, wenn Sie ähnliche Arten von Spam erhalten, da Akismet eine deutlich größere Sammlung an Vergleichsdokumenten besitzt, als Sie wohl jemals bekommen werden.

Alternative Methoden

Beide Klassifizierer in diesem Kapitel sind Beispiele für *Methoden überwachten Lernens*, also solche, die mit korrekten Ergebnissen trainiert werden und allmählich bessere Voraussagen treffen können. Das in Kapitel 4 beschriebene *künstliche neuronale Netz* zum Gewichten von Suchergebnissen war ein anderes Beispiel überwachten Lernens. Dieses neuronale Netz kann so angepasst werden, dass es die Probleme aus diesem Kapitel behandelt, indem es die Merkmale als Eingabe und die möglichen Klassifikationen als Ausgabe nutzt. Ebenso können *Support-Vektor-Maschinen*, die in Kapitel 9 beschrieben werden, auf die gleichen Probleme angewandt werden.

Bayes-Klassifizierer werden deshalb häufig für die Dokumentenklassifizierung eingesetzt, weil sie deutlich weniger Rechenleistung brauchen als andere Methoden. Eine E-Mail kann Hunderte oder Tausende von Wörtern enthalten, und das reine Anpassen der Zähler braucht deutlich weniger Speicher und Prozessorleistung, als das Trainieren eines neuronalen Netzes dieser Größe erfordern würde – wie man sieht, geht es sogar komplett ohne Datenbank. Je nach erforderlicher Geschwindigkeit beim Trainieren und Abfragen und je nach Umgebung, in der das Ganze läuft, kann ein neuronales Netz eine mögliche Alternative sein. Die Komplexität eines neuronalen Netzes sorgt allerdings auch für eine schlechtere Interpretationsmöglichkeit der Ergebnisse. In diesem Kapitel konnten Sie sich die Wortwahrscheinlichkeiten anschauen und genau erkennen, wie sehr sie zum Gesamtergebnis beitragen, während die Verbindungsgewichtungen zwischen den Neuronen in einem Netz keine genauso einfache Interpretation zulassen.

Andererseits haben neuronale Netze und Support-Vektor-Maschinen einen großen Vorteil gegenüber den in diesem Kapitel präsentierten Klassifizierern: sie können komplexere Beziehungen zwischen den Eingabemerkmalen erfassen. In einem Bayes-Klassifizierer hat jedes Merkmal eine Wahrscheinlichkeit für jede Kategorie, und Sie kombinieren die Wahrscheinlichkeiten, um eine Gesamtwahrscheinlichkeit zu erhalten. In einem neuronalen Netz kann sich die Wahrscheinlichkeit eines

e-bol.net

Merkmals abhängig vom Vorhandensein anderer Merkmale ändern. Es könnte ja sein, dass Sie versuchen, Online-Casinos abzuwehren, aber Interesse an Pferdewetten haben. In dem Fall ist das Wort »casino« schlecht, solange es nicht zusammen mit »horse« in einer E-Mail vorkommt. Naive Bayes-Klassifizierer können diese Zusammenhänge nicht abbilden – im Gegensatz zu neuronalen Netzen.

Übungen

1. *Angenommene Wahrscheinlichkeiten variieren*. Passen Sie die Klasse `classifier` so an, dass sie unterschiedliche angenommene Wahrscheinlichkeiten für unterschiedliche Merkmale ermöglicht. Ändern Sie die Methode `init`, sodass sie einen anderen Klassifizierer nutzt und mit einer besseren Schätzung als 0,5 für die angenommenen Wahrscheinlichkeiten startet.
2. *Pr(Dokument) berechnen*. Im naiven Bayes-Klassifizierer wurde die Berechnung von *Pr(Dokument)* übergangen, weil sie zum Vergleichen der Wahrscheinlichkeiten nicht notwendig war. In Fällen, in denen die Merkmale unabhängig voneinander sind, kann sie aber genutzt werden, um die Gesamtwahrscheinlichkeit zu ermitteln. Wie würden Sie *Pr(Dokument)* berechnen?
3. *Ein POP3-E-Mail-Filter*. Python bringt schon eine Bibliothek *poplib* mit, die genutzt werden kann, um E-Mails herunterzuladen. Schreiben Sie ein Skript, das E-Mails vom Server herunterlädt und versucht, sie zu klassifizieren. Welche sind die verschiedenen Eigenschaften einer E-Mail, und wie würden Sie eine Funktion zum Extrahieren von Merkmalen schreiben, die daraus einen Vorteil zieht?
4. *Anpassbare Phrasenlänge*. Dieses Kapitel hat Ihnen gezeigt, wie Sie Wortpaare und einzelne Wörter extrahieren können. Erweitern Sie die Merkmalsextrahierung, sodass man konfigurieren kann, wie viele Wörter ein einzelnes Merkmal bilden sollen.
5. *IP-Adressen erhalten*. IP-Adressen, Telefonnummern und andere Zahlen können dabei helfen, Spam zu identifizieren. Passen Sie die Funktion zur Extraktion von Merkmalen so an, dass sie diese Elemente als Merkmale zurückgibt. (IP-Adressen haben Punkte zwischen ihren Zahlen, aber Sie müssen trotzdem weiterhin die Punkte am Ende von Sätzen loswerden.)
6. *Weitere virtuelle Merkmale*. Es gibt viele virtuelle Merkmale wie UPPERCASE, die beim Klassifizieren von Dokumenten nützlich sein können. Dass Dokumente extrem groß sind oder besonders viele besonders lange Wörter enthalten, können ebenfalls Hinweise sein. Implementieren Sie dies als Merkmale. Gibt es noch andere?
7. *Neuronales Netz als Klassifizierer*. Passen Sie das neuronale Netz aus Kapitel 4 so an, dass es zum Klassifizieren von Dokumenten genutzt werden kann. Wie

e-bol.net

unterscheiden sich die Ergebnisse? Schreiben Sie ein Programm, das Dokumente viele Tausend Male klassifiziert und mit ihnen trainiert. Ermitteln Sie, wie lange es mit beiden Algorithmen dauert. Wie sind die Unterschiede?

e-bol.net

KAPITEL 7

Modellieren mit Entscheidungsbäumen

Sie haben nun ein paar verschiedene automatische Klassifizierer kennengelernt. Dieses Kapitel wird darauf aufbauen und eine sehr nützliche Methode namens *Entscheidungsbaum-Lernen* vorstellen. Anders als die meisten anderen Klassifizierer lassen sich die Modelle, die durch Entscheidungsbäume erstellt werden, leicht interpretieren – die Liste mit Zahlen bei einem Bayes-Klassifizierer teilt Ihnen zwar mit, wie wichtig jedes Wort ist, aber Sie müssen trotzdem erst Berechnungen durchführen, um zu erfahren, wie das Ergebnis sein wird. Ein neuronales Netz ist noch schwieriger zu interpretieren, da die Gewichtung der Verbindung zwischen zwei Neuronen für sich allein nur sehr wenig Bedeutung hat. Aber Sie können den Begründungsprozess eines Entscheidungsbaums durch schlichtes Betrachten verstehen, und Sie können ihn sogar in eine einfache Abfolge von if-then-Anweisungen umwandeln.

Dieses Kapitel wird drei verschiedene Beispiele vorstellen, bei denen Entscheidungsbäume eine Rolle spielen. Das erste zeigt, wie man voraussagt, welche Benutzer einer Site sehr wahrscheinlich für einen Premium-Zugriff zahlen werden. Viele Onlineanwendungen, die durch ein Abonnement oder abhängig von der Benutzung finanziert werden, geben den Anwendern die Möglichkeit, sie auszutesten, bevor sie etwas bezahlen müssen. Bei einem Abo-Modell bietet die Site meist einen zeitlich begrenzten, kostenlosen Zugang oder eine freie Version, die weniger Features besitzt. Sites, die abhängig von der Benutzung zu bezahlen sind, stellen stattdessen eventuell eine kostenlose Session oder etwas Ähnliches bereit.

Die anderen Beispiele, die danach im Kapitel behandelt werden, nutzen Entscheidungsbäume, um Immobilienpreise und eine »Hotness« zu modellieren.

Anmeldungen vorhersagen

Wenn eine Site mit hohen Benutzerzahlen auf eine neue Anwendung verlinkt, die kostenlose, aber auch kostenpflichtige Konten anbietet, kann es passieren, dass

e-bol.net

diese Anwendung Tausende neuer Benutzer gewinnt. Viele dieser Benutzer sind einfach neugierig und eigentlich gar nicht auf der Suche nach genau so einer Art von Anwendung, daher ist es sehr unwahrscheinlich, dass sie bezahlende Benutzer werden. Das erschwert es, die echten potenziellen Kunden zu erkennen und zu betreuen, daher tendieren viele Sites dazu, Massen-E-Mails an alle zu verschicken, die sich angemeldet haben, anstatt zielgerichteter vorzugehen.

Für eine planvollere Vorgehensweise wäre es hilfreich, die Wahrscheinlichkeit vorhersagen zu können, dass ein Benutzer auch ein zahlender Benutzer wird. Sie wissen bisher, dass Sie einen Bayes-Klassifizierer oder ein neuronales Netz dafür nutzen können. Aber in diesem Fall ist Klarheit sehr wichtig – wenn Sie die Faktoren kennen, die dafür sorgen, dass ein Benutzer auch ein Kunde wird, können Sie diese Informationen nutzen, um eine Werbestrategie zu unterstützen, bestimmte Aspekte der Site besser zugänglich zu machen oder um andere Strategien anzuwenden, die die Anzahl der zahlenden Kunden erhöhen.

Stellen Sie sich für dieses Beispiel eine Onlineanwendung vor, die einen kostenlosen Testzugang anbietet. Benutzer melden sich für den Testzugang an und verwenden die Site eine gewisse Zeit, nach der sie sich entscheiden können, zu einem Basic- oder Premium-Service zu wechseln. Wenn sich Benutzer für den Testzugang anmelden, werden Informationen über sie gesammelt, und am Ende des Testzeitraums vermerken die Sitebesitzer, welche Benutzer sich dazu entschieden haben, zahlende Anwender zu werden.

Damit die Anwender nicht genervt werden und sich so schnell wie möglich anmelden, stellt die Site dabei nicht sehr viele Fragen über sie – stattdessen sammelt sie Informationen aus den Serverlogs: zum Beispiel die Site, die auf sie verwiesen hat, der geografische Standort, die Anzahl an Seiten, die vor der Anmeldung angeschaut wurden, und so weiter.

Wenn Sie die Daten sammeln und in eine Tabelle stecken, sieht diese vielleicht so aus wie Tabelle 7-1.

Tabelle 7-1: Benutzerverhalten und letztendliche Servicewahl für eine Website

Referrer	Standort	FAQs gelesen?	Seiten angeschaut	Gewählter Service
Slashdot	USA	Yes	18	None
Google	France	Yes	23	Premium
Digg	USA	Yes	24	Basic
Kiwitobes	France	Yes	23	Basic
Google	UK	No	21	Premium
(direct)	New Zealand	No	12	None
(direct)	UK	No	21	Basic
Google	USA	No	24	Premium

e-bol.net

Tabelle 7-1: Benutzerverhalten und letztendliche Servicewahl für eine Website (Fortsetzung)

Referrer	Standort	FAQs gelesen?	Seiten angeschaut	Gewählter Service
Slashdot	France	Yes	19	None
Digg	USA	No	18	None
Google	UK	No	18	None
Kiwitobes	UK	No	19	None
Digg	New Zealand	Yes	12	Basic
Google	UK	Yes	18	Basic
Kiwitobes	France	Yes	19	Basic

Packen Sie die Daten in eine Liste mit Zeilen, wobei jede Zeile eine Liste mit Spalten ist. Die letzte Spalte in jeder Zeile gibt an, ob sich der Benutzer angemeldet hat oder nicht – diese Service-Spalte ist der Wert, den Sie vorhersagen wollen. Erstellen Sie eine neue Datei namens *treepredict.py*, die Sie in diesem Kapitel nutzen werden. Wenn Sie die Daten per Hand hinzufügen wollen, geben Sie den folgenden Code in der Datei ein:

```
my_data=[['slashdot','USA','yes',18,'None'],
        ['google','France','yes',23,'Premium'],
        ['digg','USA','yes',24,'Basic'],
        ['kiwitobes','France','yes',23,'Basic'],
        ['google','UK','no',21,'Premium'],
        ['(direct)','New Zealand','no',12,'None'],
        ['(direct)','UK','no',21,'Basic'],
        ['google','USA','no',24,'Premium'],
        ['slashdot','France','yes',19,'None'],
        ['digg','USA','no',18,'None'],
        ['google','UK','no',18,'None'],
        ['kiwitobes','UK','no',19,'None'],
        ['digg','New Zealand','yes',12,'Basic'],
        ['slashdot','UK','no',21,'None'],
        ['google','UK','yes',18,'Basic'],
        ['kiwitobes','France','yes',19,'Basic']]
```

Wollen Sie stattdessen die Daten herunterladen, finden Sie sie unter *http://kiwitobes.com/tree/decision_tree_example.txt*.

Um die Daten zu laden, fügen Sie die folgenden Zeilen am Anfang von *treepredict.py* ein:

```
my_data=[line.split('\t') for line in file('decision_tree_example.txt')]
```

Sie haben nun Informationen über den Standort der Benutzer, die Site, von der sie gekommen sind, und die Zeit, die sie auf Ihrer Site verbracht haben, bevor sie sich anmeldeten. Sie brauchen nun nur noch eine Möglichkeit, die Service-Spalte mit einer guten Abschätzung zu füllen.

e-bol.net

Entscheidungsbäume

Entscheidungsbäume sind eine der einfacheren Formen des maschinellen Lernens. Es handelt sich dabei um eine vollständig transparente Methode, Beobachtungen zu klassifizieren, die nach dem Training wie eine Serie von if-then-Anweisungen aussieht, die in einem Baum angeordnet wurde. Abbildung 7-1 zeigt ein Beispiel eines Entscheidungsbaums zum Klassifizieren von Früchten.

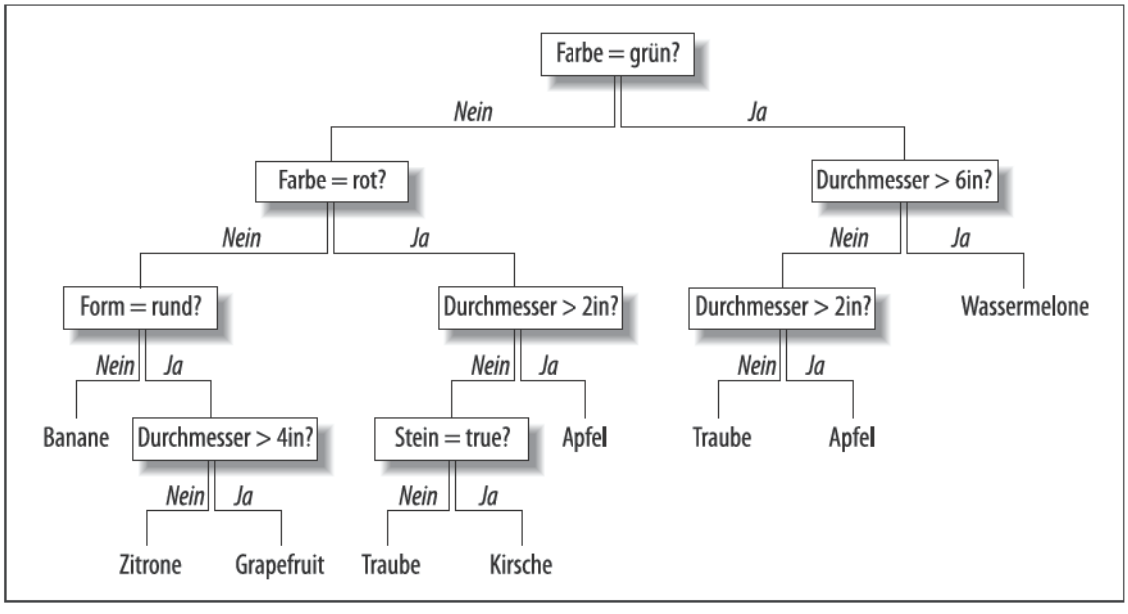

Abbildung 7-1: Beispiel-Entscheidungsbaum

Wenn Sie einmal einen Entscheidungsbaum haben, ist es recht einfach zu erkennen, wie er seine Entscheidungen trifft. Folgen Sie einfach dem Pfad den Baum hinab, der jede Frage korrekt beantwortet – dann erhalten Sie schließlich eine Antwort. Wandert man vom letzten Knoten wieder zurück nach oben, erhält man eine Begründung für die Klassifikation.

Dieses Kapitel wird zeigen, wie man einen Entscheidungsbaum repräsentieren kann, und Code vorstellen, der den Baum aus echten Daten aufbaut und der neue Beobachtungen klassifiziert. Der erste Schritt ist, eine Repräsentation eines Baums zu erzeugen. Erstellen Sie eine neue Klasse namens `decisionnode`, die jeden Knoten im Baum repräsentiert:

```
class decisionnode:
  def __init__(self,col=-1,value=None,results=None,tb=None,fb=None):
    self.col=col
    self.value=value
    self.results=results
    self.tb=tb
    self.fb=fb
```

e-bol.net

Jeder Knoten hat fünf Instanzvariablen, die alle in der Initialisierungsmethode gesetzt werden:

- `col` ist der Spaltenindex des zu testenden Kriteriums.
- `value` ist der Wert, den die Spalte haben muss, damit ein wahres Ergebnis herauskommt.
- `tb` und `fb` sind vom Typ `decisionnode` und stellen die nächsten Knoten im Baum dar, die angesprungen werden, wenn das Ergebnis `true` beziehungsweise `false` ist.
- `results` speichert ein Dictionary mit Ergebnissen für diesen Zweig ab. Dabei steht dort außer bei Endpunkten immer `None`.

Die Funktion, die einen Baum erstellt, liefert den Wurzelknoten zurück, der dann über seine True- oder False-Zweige durchlaufen werden kann, bis ein Knoten mit einem Ergebnis erreicht wird.

Trainieren des Baums

Dieses Kapitel verwendet einen Algorithmus namens *CART* (Classification and Regression Trees). Um den Entscheidungsbaum aufzubauen, erstellt der Algorithmus zunächst einen Wurzelknoten. Nachdem er alle Beobachtungen in der Tabelle begutachtet hat, wählt er die beste Variable zum Aufteilen der Daten aus. Dazu schaut er sich die verschiedenen Variablen an und entscheidet, welche Bedingung (zum Beispiel: »Hat der Benutzer die FAQ gelesen?«) die Ergebnisse (für welchen Service sich der Benutzer entschieden hat) so unterteilt, dass man leichter abschätzen kann, was der Benutzer tun wird.

`divideset` ist eine Funktion, die die Zeilen in Abhängigkeit von den Daten einer bestimmten Spalte in zwei Sets aufteilt. Diese Funktion erhält eine Liste mit Zeilen, eine Spaltennummer und einen Wert, an dem die Spalte unterteilt werden soll. Im Fall von »FAQ gelesen?« sind die möglichen Werte Ja und Nein, während es bei Referrer viele Möglichkeiten gibt. Dann werden zwei Listen mit Zeilen zurückgegebene: Die erste enthält die Zeilen, in denen die Daten in der angegebenen Spalte dem Wert entsprechen, die zweite enthält dagegen die Zeilen, in denen das nicht der Fall war.

```
# Teilt ein Set aufgrund einer bestimmten Spalte ein. Kann numerische
# oder Nominal-Werte verwenden.
def divideset(rows,column,value):
   # Funktion, um uns zu sagen, ob eine Zeile in der
   # ersten Gruppe (true) oder in der zweiten Gruppe (false) ist.
   split_function=None
   if isinstance(value,int) or isinstance(value,float):
      split_function=lambda row:row[column]>=value
   else:
      split_function=lambda row:row[column]==value
```

e-bol.net

```
    # Aufteilen und Zurückgeben der Zeilen in zwei Sets.
    set1=[row for row in rows if split_function(row)]
    set2=[row for row in rows if not split_function(row)]
    return (set1,set2)
```

Der Code erstellt eine Funktion zum Aufteilen der Daten namens `split_function`, die davon abhängig ist, ob die Daten numerisch sind oder nicht. Bei numerischen Werten wird `true` zurückgegeben, wenn der aktuelle Wert größer als `value` ist. Bei nicht numerischen Werten bestimmt `split_function` einfach, ob der Wert der Spalte der gleiche ist wie `value`. Die Funktion wird dann genutzt, um die Daten in zwei Sets aufzuteilen – eines, bei der `split_function` den Wert `true` zurückliefert, und eines, bei der `false` das Ergebnis ist.

Starten Sie eine Python-Session und versuchen Sie, die Ergebnisse anhand der Spalten »FAQ gelesen?« zu unterteilen:

```
$ python
>>> import treepredict
>>> treepredict.divideset(treepredict.my_data,2,'yes')
([['slashdot', 'USA', 'yes', 18, 'None'], ['google', 'France', 'yes', 23,
'Premium'],...]]
[['google', 'UK', 'no', 21, 'Premium'], ['(direct)', 'New Zealand', 'no', 12,
'None'],...])
```

Tabelle 7-2 zeigt die Aufteilung.

Tabelle 7-2: Ergebnisse basierend auf dem Wert der Spalten »FAQ gelesen?«

True	False
None	Premium
Premium	None
Basic	Basic
Basic	Premium
None	None
Basic	None
Basic	None

Das sieht nicht nach einer geeigneten Variablen aus, um das Ergebnis an dieser Stelle aufzuteilen, da beide Seiten ziemlich vermischt zu sein scheinen. Wir brauchen also eine Möglichkeit, die beste Variable auszuwählen.

Die beste Aufteilung wählen

Unsere eher flüchtige Beobachtung, dass die gewählte Variable nicht sehr gut sei, mag korrekt sein, aber um per Software auszuwählen, welche Variable die richtige ist, brauchen Sie eine Möglichkeit zu messen, wie gemischt ein Set ist. Denn Sie wollen ja die Variable finden, die dafür sorgt, dass die zwei Sets so wenig gemischt

e-bol.net

sind wie möglich. Die erste Funktion, die Sie benötigen, ist eine, die die Ergebnisse in jedem Set zählt. Fügen Sie diesen Code der Datei *treepredict.py* hinzu:

```
# Zählen der möglichen Ergebnisse (die letzte Spalte in jeder Zeile
# ist das Ergebnis).
def uniquecounts(rows):
   results={}
   for row in rows:
      # Das Ergebnis steht in der letzten Spalte.
      r=row[len(row)-1]
      if r not in results: results[r]=0
      results[r]+=1
   return results
```

`uniquecounts` findet alle verschiedenen möglichen Ergebnisse und liefert sie als ein Dictionary zurück, das angibt, wie oft jedes Ergebnis vorkommt. Dies wird von anderen Funktionen genutzt, um zu berechnen, wie gemischt ein Set ist. Es gibt ein paar unterschiedliche Metriken, um dies zu messen, wobei wir hier zwei vorstellen wollen: den Gini-Index und die Entropie.

Gini-Index

Der *Gini-Index* (*Gini Impurity*) ist die erwartete Fehlerrate, wenn eines der Ergebnisse aus einem Set zufällig auf eines der Elemente im Set angewandt wird. Wenn jedes Element im Set aus der gleichen Kategorie kommt, wird die Abschätzung immer korrekt sein, und die Fehlerrate beträgt dann 0. Wenn es vier mögliche Ergebnisse gibt, die in der Gruppe gleich verteilt sind, gibt es eine Chance von 75 %, dass die Abschätzung falsch ist, daher ist die Fehlerrate dann 0,75.

Die Funktion für den Gini-Index sieht so aus:

```
# Wahrscheinlichkeit, dass ein zufällig gewähltes Element
# in der falschen Kategorie ist.
def giniimpurity(rows):
  total=len(rows)
  counts=uniquecounts(rows)
  imp=0
  for k1 in counts:
    p1=float(counts[k1])/total
    for k2 in counts:
      if k1==k2: continue
      p2=float(counts[k2])/total
      imp+=p1*p2
  return imp
```

Diese Funktion berechnet die Wahrscheinlichkeit jedes möglichen Ergebnisses, indem sie die Anzahl der Vorkommen dieses Ergebnisses durch die Gesamtzahl an Zeilen im Set teilt. Dann addiert es die Produkte all dieser Wahrscheinlichkeiten. Damit erhält man die Gesamtwahrscheinlichkeit, dass eine Zeile zufällig dem falschen Ergebnis zugewiesen würde. Je größer diese Wahrscheinlichkeit ist, desto

e-bol.net

schlechter ist die Aufteilung. Eine Wahrscheinlichkeit von null ist großartig, da Sie damit wissen, dass alles schon im richtigen Set ist.

Entropie

Entropie ist in der Informationstheorie die Größe der Unordnung in einem Set – einfach gesagt also, wie gemischt ein Set ist. Fügen Sie diese Funktion der Datei *treepredict.py* hinzu:

```
# Entropie ist die Summe von p(x)log(p(x)) für alle
# möglichen unterschiedlichen Ergebnisse.
def entropy(rows):
   from math import log
   log2=lambda x:log(x)/log(2)
   results=uniquecounts(rows)
   # Berechnen der Entropie.
   ent=0.0
   for r in results.keys( ):
      p=float(results[r])/len(rows)
      ent=ent-p*log2(p)
   return ent
```

Die Entropie-Funktion berechnet die Häufigkeit jedes Elements (die Anzahl der Vorkommen geteilt durch die Gesamtanzahl an Zeilen) und wendet diese Formeln an:

p(i) = Häufigkeit(Ergebnis) = count(Ergebnis) / count(alle Zeilen)
Entropie = Summe von p(i) x log(p(i)) für alle Ergebnisse

Das ist ein Maß dafür, wie unterschiedlich die Ergebnisse untereinander sind. Wenn sie immer gleich sind (Sie also zum Beispiel wirklich Glück hatten und jeder ein Premium-Abo abgeschlossen hat), ist die Entropie 0. Je gemischter die Gruppen sind, desto größer ist ihre Entropie. Unser Ziel beim Aufteilen der Daten in zwei neue Gruppen ist es, die Entropie zu reduzieren.

Testen Sie den Gini-Index und die Entropie in Ihrer Python-Session einmal aus:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
>>> treepredict.giniimpurity(treepredict.my_data)
0.6328125
>>> treepredict.entropy(treepredict.my_data)
1.5052408149441479
>>> set1,set2=treepredict.divideset(treepredict.my_data,2,'yes')
>>> treepredict.entropy(set1)
1.2987949406953985
>>> treepredict.giniimpurity(set1)
0.53125
```

Der Hauptunterschied zwischen der Entropie und dem Gini-Index ist, dass die Entropie ihr Maximum langsamer erreicht. Konsequenterweise tendiert sie dazu,

e-bol.net

gemischte Sets etwas härter zu bestrafen. Im Rest dieses Kapitels werden wir die Entropie als Metrik nutzen, da sie häufiger genutzt wird, aber es ist einfach, sie durch den Gini-Index zu ersetzen.

Rekursiver Aufbau des Baums

Um herauszufinden, wie gut ein Attribut ist, berechnet der Algorithmus zunächst die Entropie der Gesamtgruppe. Dann teilt er die Gruppe an den möglichen Werten jedes Attributs auf und berechnet die Entropie der beiden neuen Gruppen. Um herauszufinden, welches Attribut das beste zur Unterteilung ist, wird der *Informationsgewinn* (*Information Gain*) berechnet. Der Informationsgewinn ist der Unterschied zwischen der aktuellen Entropie und dem gewichteten Durchschnitt der Entropien der beiden neuen Gruppen. Der Algorithmus berechnet den Informationsgewinn für jedes Attribut und wählt das mit dem höchsten Gewinn aus.

Nachdem die Bedingung für den Wurzelknoten ausgewählt wurde, erstellt der Algorithmus zwei Zweige, die dem Ergebnis `true` oder `false` für diese Bedingung entsprechen (siehe Abbildung 7-2).

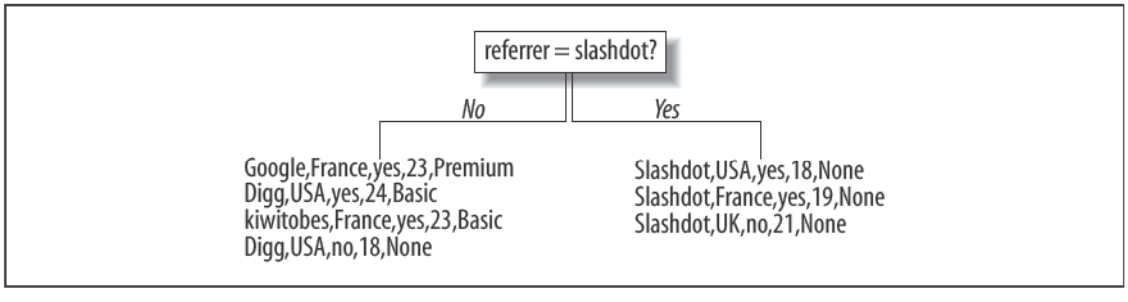

Abbildung 7-2: Entscheidungsbaum nach einer ersten Unterteilung

Die Beobachtungen sind nun unterteilt in solche, die die Bedingung erfüllen, und solche, die dies nicht tun. Für jeden Zweig bestimmt der Algorithmus nun, ob er weiter unterteilt werden kann oder ob eine stabile Schlussfolgerung erreicht wurde. Wenn einer der neuen Zweige unterteilt werden kann, wird die gleiche Methode wie oben dazu genutzt, die zu verwendende Variable zu ermitteln. Die zweite Aufteilung ist in Abbildung 7-3 zu sehen.

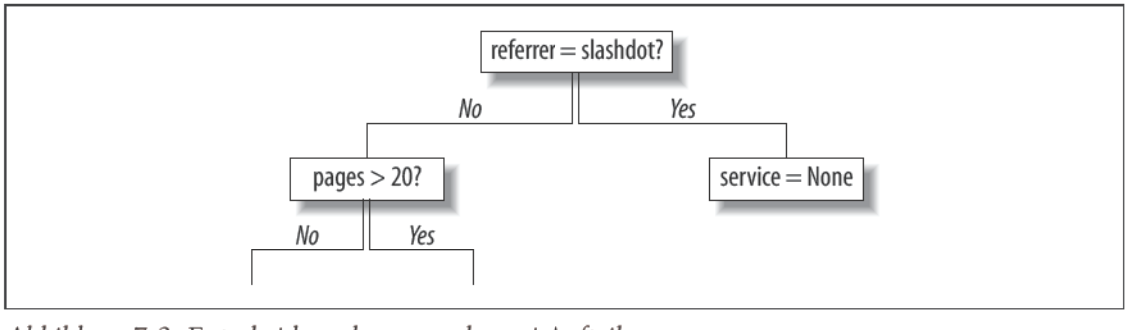

Abbildung 7-3: Entscheidungsbaum nach zwei Aufteilungen

e-bol.net

Die Zweige teilen sich weiter auf und ergeben so einen Baum, bei dem das beste Attribut für jeden neuen Knoten ermittelt wird. Ein Zweig endet ohne weitere Aufteilung, wenn der Informationsgewinn beim Aufteilen eines Knotens nicht größer als null ist.

Erstellen Sie eine neue Funktion namens `buildtree` in *treepredict.py*. Dies ist eine rekursive Funktion, die den Baum aufbaut, indem sie für das aktuelle Set das beste Unterteilungskriterium findet:

```
def buildtree(rows,scoref=entropy):
  if len(rows)==0: return decisionnode( )
  current_score=scoref(rows)

  # Variablen zum Bestimmen des besten Kriteriums.
  best_gain=0.0
  best_criteria=None
  best_sets=None

  column_count=len(rows[0])-1
  for col in range(0,column_count):
    # Erstellen der Liste mit unterschiedlichen Werten in
    # dieser Spalte.
    column_values={}
    for row in rows:
       column_values[row[col]]=1
    # Versuch, die Zeilen für jeden Wert in dieser Spalte
    # aufzuteilen.
    for value in column_values.keys( ):
      (set1,set2)=divideset(rows,col,value)

      # Informationsgewinn
      p=float(len(set1))/len(rows)
      gain=current_score-p*scoref(set1)-(1-p)*scoref(set2)
      if gain>best_gain and len(set1)>0 and len(set2)>0:
        best_gain=gain
        best_criteria=(col,value)
        best_sets=(set1,set2)
  # Erstellen der Zweige.
  if best_gain>0:
    trueBranch=buildtree(best_sets[0])
    falseBranch=buildtree(best_sets[1])
    return decisionnode(col=best_criteria[0],value=best_criteria[1],
                        tb=trueBranch,fb=falseBranch)
  else:
    return decisionnode(results=uniquecounts(rows))
```

Diese Funktion wird zunächst mit der Liste der Zeilen aufgerufen. Dabei durchläuft sie jede Spalte (außer der letzten, in der das Ergebnis steht), findet jeden möglichen Wert für diese Spalte und teilt die Datenmenge in zwei neue Untersets auf. Dann berechnet sie den gewichteten Durchschnitt der Entropie für jedes Paar Untermengen, indem sie die Entropie jeder Menge mit dem Anteil der Elemente multipliziert,

e-bol.net

die in der Menge vorhanden sind, wobei sich die Funktion merkt, welches Paar die kleinste Entropie hat.

Wenn das beste Paar der Untermengen keinen kleineren gewichteten Durchschnitt der Entropie als die aktuelle Menge hat, endet der Zweig, und die möglichen Ergebnisse werden zusammen mit ihrer Häufigkeit gespeichert. Ansonsten wird `buildtree` für jede Menge aufgerufen, die dann dem Baum hinzugefügt werden. Die Ergebnisse der Aufrufe für jede Untermenge werden an die True- und False-Zweige des Knotens gehängt, um schließlich so den gesamten Baum aufzubauen.

Jetzt können Sie endlich den Algorithmus auf die ursprüngliche Datenmenge anwenden. Der Code ist flexibel genug, um sowohl mit Text- als auch mit numerischen Daten umgehen zu können. Er geht zudem davon aus, dass die letzte Spalte den Zielwert enthält, sodass Sie einfach die Datenzeilen übergeben können, um den Baum aufzubauen:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
>>> tree=treepredict.buildtree(treepredict.my_data)
```

`tree` enthält nun einen trainierten Entscheidungsbaum. Sie werden gleich lernen, wie Sie den Baum betrachten können, und später noch erfahren, wie Sie mit ihm Voraussagen treffen können.

Anzeigen des Baums

Jetzt haben Sie einen Baum, aber was sollen Sie mit ihm anstellen? Nun, mit Sicherheit werden Sie ihn sich anschauen wollen. `printtree` ist eine einfache Funktion, die den Baum als Text ausgibt. Die Anzeige ist nicht sehr hübsch, aber eine einfache Möglichkeit, kleine Bäume anzuzeigen:

```
def printtree(tree,indent=''):
   # Ist dies ein Endknoten?
   if tree.results!=None:
      print str(tree.results)
   else:
      # Ausgeben der Kriterien.
      print str(tree.col)+':'+str(tree.value)+'? '

      # Ausgeben der Zweige.
      print indent+'T->',
      printtree(tree.tb,indent+'  ')
      print indent+'F->',
      printtree(tree.fb,indent+'  ')
```

Dies ist wieder eine rekursive Funktion. Sie erwartet einen Baum, der von `buildtree` zurückgegeben wurde, und arbeitet sich an ihm nach unten. Dabei weiß sie, dass das Ende eines Zweigs erreicht wurde, wenn sie am Knoten mit `results` angekommen ist. Bis dahin gibt die Funktion die Kriterien für die True- und False-Zweige

e-bol.net

aus und ruft für jeden von ihnen `printtree` auf, wobei jedes Mal der String zum Einrücken verlängert wird.

Rufen Sie diese Funktion mit dem Baum auf, den Sie gerade gebaut haben, erhalten Sie so etwas wie das Folgende:

```
>>> reload(treepredict)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> 0:slashdot?
  T-> {'None': 3}
  F-> 2:yes?
    T-> {'Basic': 4}
    F-> 3:21?
      T-> {'Basic': 1}
      F-> {'None': 3}
```

Dies ist eine Darstellung des Prozesses, den der Entscheidungsbaum durchlaufen wird, wenn er versucht, eine neue Klassifikation vorzunehmen. Die Bedingung am Wurzelknoten ist: »Enthält Spalte 0 Google?« Wenn diese Bedingung erfüllt ist, wird mit dem Zweig `T->` fortgefahren, und es stellt sich heraus, dass jeder, der von Google zu dieser Site gekommen ist, ein zahlender Abonnent wird, wenn er 21 Seiten oder mehr angeschaut hat. Wurde die Bedingung nicht erfüllt, springt man zum Zweig `F->` und überprüft die Bedingung »Enthält Spalte 0 Slashdot?«. Das geht so weiter, bis ein Zweig mit einem Ergebnis erreicht wird. Wie schon erwähnt, ist die Möglichkeit, die Logik hinter dem Begründungsprozess betrachten zu können, einer der großen Vorteile von Entscheidungsbäumen.

Grafische Anzeige

Die Anzeige in Textform ist für kleine Bäume ausreichend, aber wenn sie größer werden, kann es recht schwierig sein, ihren Weg durch das Geäst zu verfolgen. Hier zeige ich Ihnen, wie Sie eine grafische Repräsentation des Baums ausgeben können, die sehr nützlich für die Bäume ist, die Sie in den nächsten Abschnitten erstellen werden.

Der Code zum Zeichnen des Baums ähnelt dem für die Dendrogramme aus Kapitel 3. Beide zeichnen einen binären Baum mit Knoten einer bestimmten Tiefe, daher benötigen Sie zunächst eine Funktion, die entscheidet, wie viel Platz ein gegebener Knoten benötigt – sowohl, was die Gesamtbreite aller seiner Kindknoten angeht, als auch die Tiefe, die angibt, wie viel vertikaler Platz für alle Zweige benötigt wird. Die Gesamtbreite eines Baums ist die kombinierte Tiefe seiner Kindzweige oder 1, wenn er keine Kindzweige hat:

e-bol.net

```
def getwidth(tree):
  if tree.tb==None and tree.fb==None: return 1
  return getwidth(tree.tb)+getwidth(tree.fb)
```

Die Tiefe eines Zweigs ist 1 plus die Gesamttiefe seines längsten Kindzweigs:

```
def getdepth(tree):
  if tree.tb==None and tree.fb==None: return 0
  return max(getdepth(tree.tb),getdepth(tree.fb))+1
```

Um den Baum dann zu zeichnen, brauchen Sie eine installierte Python Imaging Library. Sie erhalten diese Bibliothek unter *http://pythonware.com*, und in Anhang A ist beschrieben, wie man sie installiert. Fügen Sie diese `import`-Anweisung am Anfang von *treepredict.py* ein:

```
from PIL import Image,ImageDraw
```

Die Funktion `drawtree` ermittelt die benötigte Gesamtgröße und bereitet einen Canvas vor. Dann übergibt sie diesen Canvas und den obersten Knoten des Baums an `drawnode`. Fügen Sie diese Funktion der Datei *treepredict.py* hinzu:

```
def drawtree(tree,jpeg='tree.jpg'):
  w=getwidth(tree)*100
  h=getdepth(tree)*100+120

  img=Image.new('RGB',(w,h),(255,255,255))
  draw=ImageDraw.Draw(img)

  drawnode(draw,tree,w/2,20)
  img.save(jpeg,'JPEG')
```

Die Funktion `drawnode` zeichnet damit die Entscheidungsknoten des Baums. Sie arbeitet rekursiv, indem sie zunächst den aktuellen Knoten zeichnet, die Positionen der Kindknoten berechnet und dann für jeden dieser Kindknoten `drawnode` aufruft. Fügen Sie diese Funktion *treepredict.py* hinzu:

```
def drawnode(draw,tree,x,y):
  if tree.results==None:
    # Breite jedes Zweigs ermitteln.
    w1=getwidth(tree.fb)*100
    w2=getwidth(tree.tb)*100

    # Größe dieses Knotens ermitteln.
    left=x-(w1+w2)/2
    right=x+(w1+w2)/2

    # Entscheidungstext zeichnen.
    draw.text((x-20,y-10),str(tree.col)+':'+str(tree.value),(0,0,0))

    # Linien zeichnen.
    draw.line((x,y,left+w1/2,y+100),fill=(255,0,0))
    draw.line((x,y,right-w2/2,y+100),fill=(255,0,0))

    # Unterknoten zeichnen.
    drawnode(draw,tree.fb,left+w1/2,y+100)
```

e-bol.net

```
    drawnode(draw,tree.tb,right-w2/2,y+100)
  else:
    txt=' \n'.join(['%s:%d'%v for v in tree.results.items()])
    draw.text((x-20,y),txt,(0,0,0))
```

Sie können nun den ganzen Baum in Ihrer Python-Session zeichnen:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.drawtree(tree,jpeg='treeview.jpg')
```

Das sollte zu einer neuen Datei *treeview.jpg* führen, die in Abbildung 7-4 zu sehen ist.

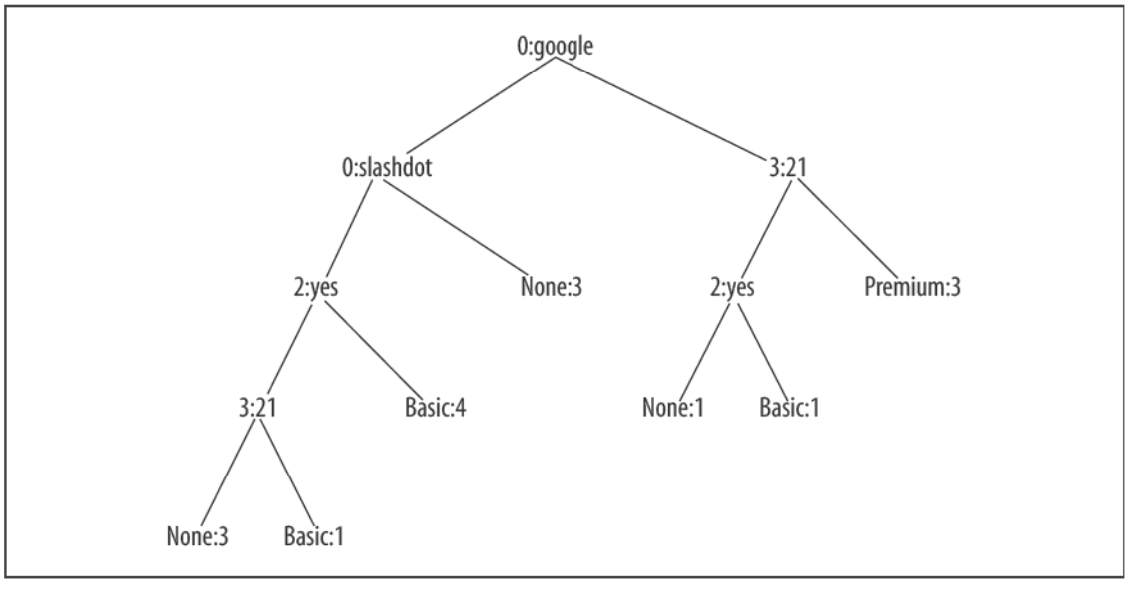

Abbildung 7-4: Entscheidungsbaum für die Vorhersage von Abonnenten

Der Code gibt die True- und False-Texte nicht mit aus, da diese die Anzeige eines größeren Baums nur vollstopfen würden. In den generierten Baumdiagrammen ist der True-Zweig immer der rechte, sodass Sie auch so dem Begründungsprozess folgen können.

Klassifikation neuer Beobachtungen

Nun brauchen Sie noch eine Funktion, die eine neue Beobachtung erwartet und sie nach Vorgabe des Entscheidungsbaums klassifiziert. Fügen Sie diese Funktion der Datei *treepredict.py* hinzu:

```
def classify(observation,tree):
  if tree.results!=None:
    return tree.results
  else:
    v=observation[tree.col]
    branch=None
```

e-bol.net

```
    if isinstance(v,int) or isinstance(v,float):
      if v>=tree.value: branch=tree.tb
      else: branch=tree.fb
    else:
      if v==tree.value: branch=tree.tb
      else: branch=tree.fb
    return classify(observation,branch)
```

Diese Funktion arbeitet sich durch den Baum auf die gleiche Weise durch, wie es `printtree` tut. Nach jedem Aufruf prüft sie, ob das Ende dieses Zweigs erreicht wurde, indem sie `results` kontrolliert. Wenn nicht, wertet sie die Beobachtung aus, um zu prüfen, ob die Spalte dem Wert entspricht. Ist das der Fall, ruft sie erneut `classify` für den True-Zweig auf; wenn nicht, wird `classify` stattdessen für den False-Zweig aufgerufen.

Jetzt können Sie `classify` aufrufen, um die Vorhersage für eine neue Beobachtung zu erhalten:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.classify(['(direct)','USA','yes',5],tree)
{'Basic': 4}
```

Sie haben nun Funktionen zum Aufbau eines Entscheidungsbaums aus einer beliebigen Datenmenge, zum Anzeigen und Interpretieren des Baums und zum Klassifizieren neuer Ergebnisse. Diese Funktionen können auf jede Datenmenge angewandt werden, die aus mehreren Zeilen besteht, die jeweils eine Reihe von Beobachtungen und ein Ergebnis enthalten.

Entscheidungsbäume in der Anwendung

Da sich Entscheidungsbäume so einfach interpretieren lassen, handelt es sich um die am weitesten verbreiteten Data-Mining-Methoden in der betriebswirtschaftlichen Analyse sowie bei der Entscheidungsfindung in der Medizin und in der Versicherungswirtschaft. Häufig wird ein Entscheidungsbaum automatisch erstellt und dann von einem Experten analysiert, um die Schlüsselfaktoren zu verstehen und den Baum danach an die Vorstellungen des Experten anzupassen. Dieser Prozess ermöglicht es dem Computer, Experten zu unterstützen und den Begründungsprozess deutlich zu machen, sodass man als Einzelperson die Qualität der Vorhersage bewerten kann.

Entscheidungsbäume wurden auf diese Art und Weise in sehr vielen Anwendungen genutzt, wie zum Beispiel beim Erstellen von Kundenprofilen, der Risikoanalyse im Finanzbereich, zum Unterstützen von Diagnosen und bei der Verkehrsplanung.

e-bol.net

Zurechtstutzen des Baums

Ein Problem beim Trainieren des Baums mit den bisher beschriebenen Methoden ist, dass er *überangepasst* (*overfitted*) werden kann – sich also zu sehr an den Trainingsdaten orientiert. Ein überangepasster Baum kann eine Antwort geben, die eindeutiger ist, als sie eigentlich sein sollte. Das kann geschehen, indem er Zweige erstellt, die die Entropie für das Trainingsset zwar ein wenig verringern, aber deren Bedingungen in Wirklichkeit rein willkürlich sind.

Da der oben beschriebene Algorithmus die Zweige so lange weiter unterteilt, bis er die Entropie nicht weiter reduzieren kann, könnte man ihn so anpassen, dass nicht mehr weiter unterteilt wird, wenn die Entropie sich nicht um einen Mindestbetrag reduziert. Diese Strategie wird häufig angewendet, aber sie hat auch einen kleinen Nachteil – es ist möglich, eine Datenmenge zu haben, bei der sich die Entropie pro Aufteilung nur wenig verringert, aber mehrere Aufteilungen sich doch deutlich bemerkbar machen. Eine Alternative wäre ein Aufbau des Baums in der oben beschriebenen Art und Weise, wobei man zum Schluss versucht, überflüssige Knoten zu entfernen. Dieser Prozess wird als *Pruning* (*Beschneiden*) bezeichnet.

Zum Pruning gehört, Knotenpaare zu überprüfen, die gemeinsame Elternknoten haben, um herauszufinden, ob eine Verschmelzung beider Knoten nur zu einer geringfügigen Steigerung der Entropie führen würde. Wenn dies der Fall ist, werden die Blätter zu einem einzelnen Knoten mit allen möglichen Ergebnissen verschmolzen. Das hilft dabei, eine Überanpassung zu vermeiden und den Baum davon abzuhalten, Vorhersagen zu treffen, die zuverlässiger erscheinen, als es die Daten wirklich hergeben.

Fügen Sie eine neue Funktion der Datei *treepredict.py* hinzu, um den Baum zu beschneiden:

```
def prune(tree,mingain):
  # Wenn die Zweige keine Blätter sind, beschneide sie.
  if tree.tb.results==None:
    prune(tree.tb,mingain)
  if tree.fb.results==None:
    prune(tree.fb,mingain)

  # Wenn beide Zweige Blätter sind, prüfe, ob sie sich
  # verbinden lassen.
  if tree.tb.results!=None and tree.fb.results!=None:
    # Aufbau einer kombinierten Datenmenge.
    tb,fb=[],[]
    for v,c in tree.tb.results.items():
      tb+=[[v]]*c
    for v,c in tree.fb.results.items():
      fb+=[[v]]*c

    # Verringerung der Entropie testen.
```

e-bol.net

```
    delta=entropy(tb+fb)-(entropy(tb)+entropy(fb)/2)

    if delta<mingain:
      # Verbinden der Zweige.
      tree.tb,tree.fb=None,None
      tree.results=uniquecounts(tb+fb)
```

Wenn diese Funktion für den Wurzelknoten aufgerufen wird, wandert sie den gesamten Baum zu den Knoten herab, die nur noch Blätterknoten als Kinder haben. Sie wird eine kombinierte Liste mit Ergebnissen aus beiden Blättern erstellen und die Entropie testen. Ist die Änderung der Entropie kleiner als der Parameter `mingain`, werden die Blätter gelöscht und ihre Ergebnisse zum Elternknoten verschoben. Der kombinierte Knoten wird dann wieder ein möglicher Kandidat zum Löschen und Verschmelzen mit einem anderen Knoten.

Probieren Sie es einmal mit Ihrem aktuellen Datenbestand aus, um herauszufinden, ob einige der Knoten verschmolzen werden:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.prune(tree,0.1)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> 0:slashdot?
  T-> {'None': 3}
  F-> 2:yes?
    T-> {'Basic': 4}
    F-> 3:21?
      T-> {'Basic': 1}
      F-> {'None': 3}
>>> treepredict.prune(tree,1.0)
>>> treepredict.printtree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> {'None': 6, 'Basic': 5}
```

Im Beispiel lassen sich die Daten recht gut aufteilen, daher wird ein Pruning mit einem vernünftigen niedrigen Schwellenwert gar nichts tun. Nur wenn der minimale Gewinn sehr hoch gesetzt wird, werden zwei der Blätter verschmolzen. Wie Sie später sehen werden, tendieren echte Daten nicht dazu, sich so gut aufteilen zu lassen, wie es hier der Fall ist, daher ist das Pruning in solchen Fällen deutlich effektiver.

e-bol.net

Umgang mit fehlenden Daten

Ein anderer Vorteil von Entscheidungsbäumen ist ihre Fähigkeit, mit fehlenden Daten umzugehen. Vielleicht fehlen in Ihrem Datenbestand bestimmte Informationen – im hier genutzten Beispiel kann beispielsweise der geografische Standort eines Benutzer nicht aus seiner IP-Adresse ermittelt werden, daher ist das Feld dann leer. Um den Entscheidungsbaum so anzupassen, dass er damit umgehen kann, müssen Sie eine andere Vorhersagefunktion implementieren.

Wenn Ihnen Daten fehlen, die benötigt werden, um zu entscheiden, welchem Zweig des Baums gefolgt werden soll, können Sie einfach *beiden* Zweigen folgen. Aber anstatt die Ergebnisse beider Seiten gleich zu werten, werden sie unterschiedlich gewichtet. Im einfachen Entscheidungsbaum hat alles eine implizierte Gewichtung von 1, was bedeutet, dass die Beobachtungen für die Wahrscheinlichkeit, dass ein Element in eine bestimmte Kategorie passt, allein entscheidend ist. Wenn Sie stattdessen mehreren Zweigen folgen, können Sie jedem Zweig eine Gewichtung verpassen, die dem Anteil an all den Zeilen entspricht, die sich auf dieser Seite befinden.

Die Funktion dafür, `mdclassify`, ist eine einfache Variante von `classify`. Fügen Sie sie *treepredict.py* hinzu:

```
def mdclassify(observation,tree):
  if tree.results!=None:
    return tree.results
  else:
    v=observation[tree.col]
    if v==None:
      tr,fr=mdclassify(observation,tree.tb),mdclassify(observation,tree.fb)
      tcount=sum(tr.values( ))
      fcount=sum(fr.values( ))
      tw=float(tcount)/(tcount+fcount)
      fw=float(fcount)/(tcount+fcount)
      result={}
      for k,v in tr.items( ): result[k]=v*tw
      for k,v in fr.items( ): result[k]=v*fw
      return result
    else:
      if isinstance(v,int) or isinstance(v,float):
        if v>=tree.value: branch=tree.tb
        else: branch=tree.fb
      else:
        if v==tree.value: branch=tree.tb
        else: branch=tree.fb
      return mdclassify(observation,branch)
```

Der einzige Unterschied ist, dass die Ergebnisse für jeden Zweig berechnet und dann am Ende mit ihrer Gewichtung kombiniert werden, wenn die wichtigen Datenelemente fehlen.

e-bol.net

Testen Sie `mdclassify` mit einer Zeile, in der eine entscheidende Information fehlt, und schauen Sie sich Ihre Ergebnisse an:

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.py'>
>>> treepredict.mdclassify(['google',None,'yes',None],tree)
{'Premium': 1.5, 'Basic': 1.5}
>>> treepredict2.mdclassify(['google','France',None,None],tree)
{'None': 0.125, 'Premium': 2.25, 'Basic': 0.125}
```

Lässt man die Seitenvariable leer, erhält man wie erwartet eine hohe Wahrscheinlichkeit für Premium und eine kleine für Basic. Lässt man die Variable »FAQ gelesen?« weg, sieht die Verteilung anders aus, wobei jede Möglichkeit am Ende danach gewichtet wird, wie viele Elemente sich auf jeder Seite befanden.

Umgang mit numerischen Ergebnissen

Bei den Beispielen für das Benutzerverhalten und den Fruchtbaum handelte es sich jeweils um Klassifikationsprobleme, da es sich bei den Ergebnissen um Kategorien und nicht um Zahlen handelt. Die restlichen Beispiele in diesem Kapitel – Immobilienpreise und die »Hotness« – sind Probleme mit numerischen Ergebnissen.

Obwohl es möglich ist, `buildtree` für eine Datenmenge mit Zahlen als Ergebnissen laufen zu lassen, werden die Ergebnisse wohl nicht sehr gut sein. Wenn alle Zahlen als unterschiedliche Kategorien behandelt werden, kann der Algorithmus die Tatsache, dass sich manche Zahlen nahe beieinander befinden und andere weit auseinanderliegen, nicht berücksichtigen – sie werden alle völlig getrennt behandelt werden. Um das zu vermeiden, wenn Sie einen Baum mit numerischen Ergebnissen haben, können Sie `variance` als Bewertungsfunktion statt der Entropie oder des Gini-Index nutzen. Fügen Sie die Funktion `variance` der Datei *treepredict.py* hinzu:

```
def variance(rows):
  if len(rows)==0: return 0
  data=[float(row[len(row)-1]) for row in rows]
  mean=sum(data)/len(data)
  variance=sum([(d-mean)**2 for d in data])/len(data)
  return variance
```

Diese Funktion ist ein möglicher Parameter für `buildtree`, und sie berechnet die statistische Varianz für eine Reihe von Zeilen. Eine geringe Varianz bedeutet, dass die Zahlen alle nahe beieinanderliegen, während eine hohe Varianz anzeigt, dass sie weit gestreut sind. Wenn man einen Baum mit der Varianz als Bewertungsfunktion aufbaut, werden Knotenkriterien ausgewählt werden, die die Zahlen so aufteilen, dass sich höhere Werte auf einer Seite und niedrige auf der anderen Seite befinden. Das Aufteilen der Daten auf diese Art und Weise verringert die Gesamtvarianz für die Zweige.

e-bol.net

Modellieren von Immobilienpreisen

Es gibt viele mögliche Anwendungen für Entscheidungsbäume, aber sie sind dann am nützlichsten, wenn es viele Variablen gibt und Sie am Begründungsprozess interessiert sind. In manchen Fällen kennen Sie bereits die Ergebnisse, und der interessante Teil ist hier das Modellieren, um zu verstehen, warum die Ergebnisse so sind, wie sie sind. Ein Bereich, in dem dies potenziell sehr interessant ist, ist das Verstehen von Preisen von Gütern, insbesondere von solchen, bei denen sich viele verschiedene Maßstäbe anlegen lassen. Dieser Abschnitt wird sich den Aufbau von Entscheidungsbäumen vornehmen, mit denen Immobilienpreise modelliert werden, da sich Häuser im Preis sehr unterscheiden können und es viele numerische und nominelle Variablen gibt, die sich leicht messen lassen.

Die Zillow-API

Zillow ist ein freier Webservice, der Immobilienpreise (in den USA) dokumentiert und diese Informationen nutzt, um Preisabschätzungen für andere Häuser zu erstellen. Er schaut sich Vergleichshäuser an und verwendet die Werte, um einen neuen Wert vorherzusagen. Das entspricht dem Vorgehen von professionellen Gutachtern. Ein Bereich einer Zillow-Webseite, der Informationen über ein Haus und seinen geschätzten Wert anzeigt, sehen Sie in Abbildung 7-5.

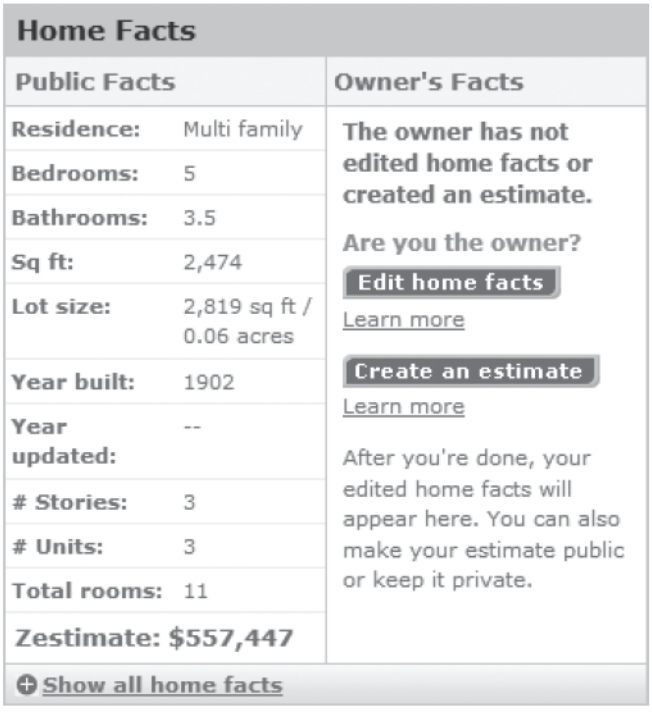

Abbildung 7-5: Screenshot von zillow.com

e-bol.net

Glücklicherweise hat Zillow auch eine API, mit der Sie Details und den Schätzwert eines Hauses abfragen können. Die Seite für die Zillow-API ist *http://www.zillow.com/howto/api/APIOverview.htm*.

Sie brauchen einen Entwicklerschlüssel, um auf die API zugreifen zu können. Diesen erhalten Sie kostenlos auf der Website. Die API selbst ist recht einfach – man ruft eine URL mit allen Suchparametern für die Abfrage auf und analysiert das zurückgegebene XML, um Details zu erhalten, wie zum Beispiel die Zimmerzahl oder den geschätzten Preis. Erstellen Sie eine neue Datei *zillow.py* und fügen Sie den folgenden Code hinzu:

```
import xml.dom.minidom
import urllib2

zwskey="X1-ZWz1chwxis15aj_9skq6"
```

Wie schon in Kapitel 5 werden Sie die *minidom*-API nutzen, um die XML-Ergebnisse Ihrer Anfragen zu parsen. Die Funktion `getaddressdata` erwartet eine Adresse und eine Stadt und baut daraus eine URL, um Zillow die entsprechenden Informationen zu entlocken. Dann werden die Ergebnisse geparst und die wichtigen Informationen extrahiert, was schließlich zu einem Ergebnis-Tupel führt. Fügen Sie diese Funktion der Datei *zillow.py* hinzu:

```
def getaddressdata(address,city):
  escad=address.replace(' ','+')

  # Aufbau der URL.
  url='http://www.zillow.com/webservice/GetDeepSearchResults.htm?'
  url+='zws-id=%s&address=%s&citystatezip=%s' % (zwskey,escad,city)

  # Parsen des XML-Ergebnisses.
  doc=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))
  code=doc.getElementsByTagName('code')[0].firstChild.data

  # Code 0 bedeutet Erfolg, ansonsten gab es einen Fehler.
  if code!='0': return None

  # Informationen über diese Eigenschaft extrahieren.
  try:
    zipcode=doc.getElementsByTagName('zipcode')[0].firstChild.data
    use=doc.getElementsByTagName('useCode')[0].firstChild.data
    year=doc.getElementsByTagName('yearBuilt')[0].firstChild.data
    bath=doc.getElementsByTagName('bathrooms')[0].firstChild.data
    bed=doc.getElementsByTagName('bedrooms')[0].firstChild.data
    rooms=doc.getElementsByTagName('totalRooms')[0].firstChild.data
    price=doc.getElementsByTagName('amount')[0].firstChild.data
  except:
    return None

  return (zipcode,use,int(year),float(bath),int(bed),int(rooms),price)
```

e-bol.net

Das von dieser Funktion zurückgegebene Tupel kann als Beobachtung in eine Liste gesteckt werden, da sich das »Ergebnis« (der Preis) am Ende befindet. Um diese Funktion dazu zu nutzen, eine vollständige Datenmenge zu erstellen, brauchen Sie eine Liste mit Adressen. Sie können diese selbst erstellen oder eine Liste mit zufällig erzeugten Adressen in Cambridge, MA unter *http://kiwitobes.com/addresslist.txt* herunterladen.

Erstellen Sie eine neue Funktion namens `getpricelist`, um diese Datei einzulesen und eine Liste mit Daten zu erzeugen:

```
def getpricelist( ):
  l1=[]
  for line in file('addresslist.txt'):
    data=getaddressdata(line.strip( ),'Cambridge,MA')
    l1.append(data)
  return l1
```

Sie können nun diese Funktionen nutzen, um eine Datenmenge zu erstellen und einen Entscheidungsbaum aufzubauen. Testen Sie sie in Ihrer Python-Session:

```
>>> import zillow
>>> housedata=zillow.getpricelist( )
>>> reload(treepredict)
>>> housetree=treepredict.buildtree(housedata,scoref=treepredict.variance)
>>> treepredict.drawtree(housetree,'housetree.jpg')
```

Eine der so generierbaren Dateien, *housetree.jpg*, ist in Abbildung 7-6 zu sehen.

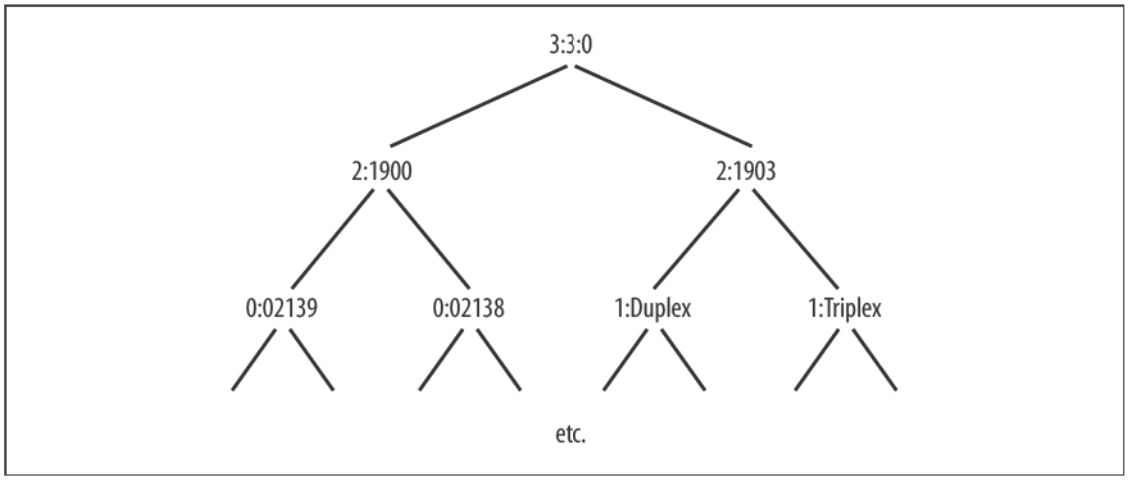

Abbildung 7-6: Entscheidungsbaum für Immobilienpreise

Wenn Sie natürlich nur daran interessiert sind, den Preis eines bestimmten Hauses zu schätzen, könnten Sie einfach die Zillow-API nutzen, um eine Schätzung zu erstellen. Interessant ist hier, dass Sie ein Modell für die Faktoren aufgebaut haben, die man beim Bestimmen von Immobilienpreisen berücksichtigt. Beachten Sie, dass sich am Wurzelknoten die Badezimmer befinden. Das bedeutet, dass sich die Varianz am meisten verringert, wenn man die Daten anhand der Anzahl der Badezim-

e-bol.net

mer aufteilt. Hauptfaktor beim Preis eines Hauses in Cambridge ist, ob es drei oder mehr Badezimmer hat (normalerweise bedeutet das, dass es sich um ein großes Mehrfamilienhaus handelt).

Der offensichtliche Nachteil eines Entscheidungsbaums ist hier, dass man Preisbereiche erstellen muss. Denn jedes Haus hat einen anderen Preis, und man muss die Preise irgendwie gruppieren, um sinnvolle Endpunkte zu erhalten. Möglicherweise funktioniert eine andere Vorhersagetechnik mit den echten Preisdaten hier besser. Kapitel 8 behandelt eine weitere Methode, um Preisabschätzungen zu geben.

Modellieren der »Hotness«

Hot or Not ist eine Site, auf der Benutzer Fotos von sich selbst hochladen können. Das ursprüngliche Konzept war, die Benutzer andere Benutzer aufgrund ihrer äußerlichen Erscheinung bewerten zu lassen und die Ergebnisse so zusammenzufassen, dass für jede Person ein Wert zwischen 1 und 10 herauskommt. Die Site hat sich seitdem zu einer Dating-Site gewandelt und bietet nun eine offene API an, die es Ihnen ermöglicht, demografische Informationen über Mitglieder zu erhalten – zusammen mit ihrer »Hotness«. Das ermöglicht einen interessanten Test für ein Entscheidungsbaum-Modell, da es eine Reihe von Eingabevariablen, eine Ausgabevariable und einen möglicherweise interessanten Begründungsprozess gibt. Die Site selbst ist zudem ein gutes Beispiel für »kollektive Intelligenz«.

Auch hier benötigen Sie einen Anwendungsschlüssel, um auf die API zuzugreifen. Sie können sich anmelden und einen Schlüssel unter *http://dev.hotornot.com/signup* erhalten.

Die API von *Hot or Not* funktioniert ähnlich wie die anderen von uns bereits benutzten APIs. Sie übergeben einfach die Parameter einer Abfrage an eine URL und parsen das zurückgegebene XML. Dazu erstellen Sie erst einmal eine neue Datei namens *hotornot.py* und tragen die `import`-Anweisungen sowie Ihren Schlüssel ein:

```
import urllib2
import xml.dom.minidom

api_key="479NUNJHETN"
```

Als Nächstes benötigen Sie eine Liste mit zufälligen Personen, um die Datenmenge aufzubauen. Glücklicherweise stellt *Hot or Not* einen API-Aufruf zur Verfügung, der eine Liste mit Personen zurückgibt, die bestimmte Kriterien erfüllen. In diesem Beispiel wird das einzige Kriterium sein, dass Personen »meet me«-Profile besitzen, da Sie nur aus solchen Profilen andere Informationen als den Standort und die Interessen erhalten können. Fügen Sie diese Funktion der Datei *hotornot.py* hinzu:

```
def getrandomratings(c):
  # URL für getRandomProfile erstellen.
```

e-bol.net

```
  url="http://services.hotornot.com/rest/?app_key=%s" % api_key
  url+="&method=Rate.getRandomProfile&retrieve_num=%d" % c
  url+="&get_rate_info=true&meet_users_only=true"

  f1=urllib2.urlopen(url).read( )

  doc=xml.dom.minidom.parseString(f1)

  emids=doc.getElementsByTagName('emid')
  ratings=doc.getElementsByTagName('rating')

  # emids und ratings in einer Liste kombinieren.
  result=[]
  for e,r in zip(emids,ratings):
    if r.firstChild!=None:
      result.append((e.firstChild.data,r.firstChild.data))
  return result
```

Nachdem Sie eine Liste mit Benutzer-IDs und Bewertungen erstellt haben, brauchen Sie eine weitere Funktion, um Informationen über diese Personen herunterzuladen – in diesem Fall Geschlecht, Alter, Standort und Schlüsselwörter. Die Nutzung aller 50 US-Bundesstaaten als mögliche Standort-Variablenwerte würde zu zu vielen Verzweigungsmöglichkeiten führen. Um die Anzahl an Möglichkeiten für den Standort zu reduzieren, können Sie die Bundesstaaten in Regionen aufteilen. Fügen Sie den folgenden Code ein, um Regionen festzulegen:

```
stateregions={'New England':['ct','mn','ma','nh','ri','vt'],
              'Mid Atlantic':['de','md','nj','ny','pa'],
              'South':['al','ak','fl','ga','ky','la','ms','mo',
                       'nc','sc','tn','va','wv'],
              'Midwest':['il','in','ia','ks','mi','ne','nd','oh','sd','wi'],
              'West':['ak','ca','co','hi','id','mt','nv','or','ut','wa','wy']}
```

Die API stellt eine Methode zum Herunterladen demografischer Daten für einzelne Personen bereit, daher läuft die Funktion `getpeopledata` einfach über alle Ergebnisse der ersten Suche und bittet die API um deren Details. Fügen Sie diese Funktion *hotornot.py* hinzu:

```
def getpeopledata(ratings):
  result=[]
  for emid,rating in ratings:
    # URL für die Methode MeetMe.getProfile.
    url="http://services.hotornot.com/rest/?app_key=%s" % api_key
    url+="&method=MeetMe.getProfile&emid=%s&get_keywords=true" % emid

    # Alle Informationen über diese Person holen.
    try:
      rating=int(float(rating)+0.5)
      doc2=xml.dom.minidom.parseString(urllib2.urlopen(url).read( ))
      gender=doc2.getElementsByTagName('gender')[0].firstChild.data
      age=doc2.getElementsByTagName('age')[0].firstChild.data
      loc=doc2.getElementsByTagName('location')[0].firstChild.data[0:2]
```

e-bol.net

```
        # Bundesstaaten in Regionen umwandeln.
        for r,s in stateregions.items( ):
          if loc in s: region=r

        if region!=None:
          result.append((gender,int(age),region,rating))
      except:
        pass
    return result
```

Sie können nun dieses Modul in Ihre Python-Session importieren und eine Datenmenge erzeugen:

```
>>> import hotornot
>>> l1=hotornot.getrandomratings(500)
>>> len(l1)
442
>>> pdata=hotornot.getpeopledata(l1)
>>> pdata[0]
(u'female', 28, 'West', 9)
```

Die Liste enthält Informationen über jeden Benutzer, wobei seine Bewertung im letzten Feld steht. Diese Datenstruktur kann direkt an die Methode `buildtree` übergeben werden, um einen Baum aufzubauen:

```
>>> hottree=treepredict.buildtree(pdata,scoref=treepredict.variance)
>>> treepredict.prune(hottree,0.5)
>>> treepredict.drawtree(hottree,'hottree.jpg')
```

Ein mögliches Ergebnis für den Baum sehen Sie in Abbildung 7-7.

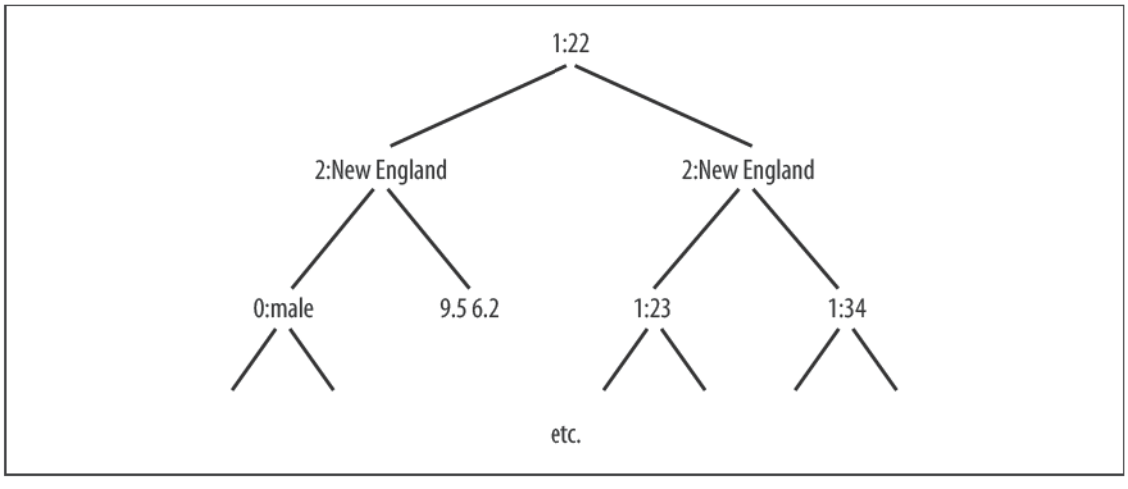

Abbildung 7-7: Entscheidungsbaum-Modell für die »Hotness«

Der zentrale Knoten, der die Daten am besten unterteilt, ist das Geschlecht. Der Rest des Baums ist dann recht kompliziert und lässt sich auch schlecht lesen. Aber Sie können ihn auf jeden Fall nutzen, um Vorhersagen über bisher noch nicht präsentierte Personen zu machen. Da der Algorithmus zudem auch fehlende Daten ver-

e-bol.net

trägt, können Sie Personen für bestimmte Variablen aggregieren. Wenn Sie zum Beispiel die »Hotness« von allen im Süden mit allen in Mid-Atlantic vergleichen wollen, geht das so:

```
>>> south=treepredict2.mdclassify((None,None,'South'),hottree)
>>> midat=treepredict2.mdclassify((None,None,'Mid Atlantic'),hottree)
>>> south[10]/sum(south.values())
0.055820815183261735
>>> midat[10]/sum(midat.values())
0.048972797320600864
```

Für diese Datenmenge gibt es ein wenig mehr besonders heiße Personen im Süden. Sie können noch andere Dinge ausprobieren, so zum Beispiel Altersgruppen einführen oder prüfen, ob Männer bessere Bewertungen bekommen als Frauen.

Wann nutzt man Entscheidungsbäume?

Der wohl größte Vorteil von Entscheidungsbäumen ist, dass sie sich leicht interpretieren lassen. Nachdem der Algorithmus für unser Beispielproblem ausgeführt wurde, haben wir nicht nur einen Baum erhalten, der Vorhersagen für neue Benutzer machen kann, sondern wir haben auch noch eine Liste mit Fragen vorliegen, die genutzt wurden, um diese Entscheidungen zu treffen. So können Sie zum Beispiel erkennen, dass Benutzer, die über Slashdot zur Site gelangt sind, niemals zahlende Abonnenten werden, während Benutzer, die von Google kommen und mindestens 20 Seiten betrachtet haben, sehr wahrscheinlich Premium-Abonnenten werden. Das ermöglicht Ihnen wiederum, eventuell Ihre Werbestrategie anzupassen und sich auf bestimmte Sites zu konzentrieren. Wir haben auch gelernt, dass bestimmte Variablen, wie zum Beispiel das Herkunftsland des Benutzers, für die Bestimmung des Ergebnisses nicht wichtig ist. Wenn sich Daten nur schwierig oder unter hohem Aufwand sammeln lassen und wir feststellen, dass sie nicht wichtig sind, können wir damit aufhören, sie zu sammeln.

Anders als bei manchen anderen Algorithmen zum maschinellen Lernen können Entscheidungsbäume sowohl mit Kategorien als auch mit numerischen Daten als Eingabewerte arbeiten. Im ersten Beispiel haben wir eine Kombination aus der Anzahl der betrachteten Seiten mit mehreren Kategorievariablen verwendet. Während zudem viele Algorithmen eine Vorverarbeitung oder Normalisierung der Daten vor der Verwendung erfordern, kann der Code in diesem Kapitel mit jeglicher Form von Daten (sofern es sich um Kategorien oder Zahlen handelt) umgehen und daraus passende Entscheidungsbäume erstellen.

Entscheidungsbäume ermöglichen auch Zuweisungen von Daten aufgrund von Wahrscheinlichkeiten. Bei manchen Problemen gibt es nicht genug Informationen, um immer korrekte Entscheidungen zu treffen – ein Entscheidungsbaum kann einen Knoten mit vielen Möglichkeiten haben, der sich nicht weiter unterteilen lässt. Der Code in diesem Kapitel liefert ein Dictionary mit der Anzahl der verschie-

e-bol.net

denen Ergebnisse zurück, und diese Information kann uns dabei helfen zu entscheiden, wie sehr wir den Ergebnissen vertrauen. Nicht alle Algorithmen können die Wahrscheinlichkeit eines unsicheren Ergebnisses abschätzen.

Aber es gibt natürlich auch Nachteile bei den Entscheidungsbaum-Algorithmen, die wir hier genutzt haben. So können sie zwar für Probleme mit nur ein paar wenigen möglichen Ergebnissen sehr effektiv sein, die Algorithmen lassen sich aber nicht sehr gut für Datensätze mit vielen Möglichkeiten nutzen. Im ersten Beispiel gab es als mögliche Ergebnisse nur Keiner, Basic und Premium. Wenn es aber stattdessen Hunderte von Ergebnissen gäbe, würde der Entscheidungsbaum sehr kompliziert werden und vermutlich schlechte Vorhersagen treffen.

Der andere große Nachteil der hier beschriebenen Entscheidungsbäume ist, dass sie zwar mit einfachen numerischen Daten umgehen, dabei aber nur Entscheidungspunkte mit »größer/kleiner als« erzeugen können. Das macht es schwierig, Daten zu klassifizieren, bei denen die Klasse durch eine komplexere Kombination aus Variablen definiert wird. Wenn sich die Ergebnisse zum Beispiel durch die Differenz zweier Variablen bestimmen, würde der Baum sehr groß werden und schnell ungenau sein.

Entscheidungsbäume sind also vermutlich keine gute Wahl für Probleme mit vielen numerischen Ein- und Ausgaben oder mit vielen komplexen Beziehungen zwischen numerischen Eingaben, wie zum Beispiel beim Interpretieren von Finanzdaten oder bei der Bildanalyse. Entscheidungsbäume sind aber sehr hilfreich bei Datenmengen mit vielen Kategoriedaten und mit numerischen Daten, die sich an einer Stelle aufteilen lassen. Diese Bäume sind dann erste Wahl, wenn das Verstehen des Entscheidungsprozesses wichtig ist – wie Sie gesehen haben, kann das Wissen um die Gründe für eine Entscheidung genauso wichtig sein wie die Entscheidung selbst.

Übungen

1. *Ergebniswahrscheinlichkeiten*. Momentan geben die Funktionen `classify` und `mdclassify` ihre Ergebnisse als Anzahl von Zeilen aus. Passen Sie sie so an, dass sie die Wahrscheinlichkeit dafür ausgeben, dass das Ergebnis in eine der Kategorien passt.
2. *Fehlende Datenbereiche*. `mdclassify` ermöglicht die Verwendung von »None«, um einen fehlenden Wert anzugeben. Bei numerischen Ergebnissen kann es sein, dass der Wert nicht vollständig unbekannt, sondern lediglich ein Bereich bekannt ist. Passen Sie `mdclassify` so an, dass auch Tupel, wie zum Beispiel (20,25), für einen Wert möglich sind, und verfolgen Sie dann bei Bedarf beide Zweige.
3. *Abbruch*. Anstatt den Baum zu beschneiden, kann `buildtree` auch einfach mit dem Aufteilen in Zweige aufhören, wenn sich die Entropie nicht mehr ausrei-

e-bol.net

chend verringert. Das mag in manchen Fällen nicht ideal sein, aber es erspart einen Zusatzschritt. Passen Sie `buildtree` so an, dass ein Parameter für den minimalen Entropiegewinn übergeben werden kann, und beenden Sie die Aufteilung, wenn diese Bedingung nicht erfüllt wurde.

4. *Erstellen des Baums mit fehlenden Daten*. Sie haben eine Funktion geschrieben, die eine Zeile mit fehlenden Daten klassifizieren kann. Was ist aber, wenn es im Trainingsset fehlende Daten gibt? Passen Sie `buildtree` so an, dass es auf fehlende Daten prüft und in Fällen, in denen es nicht möglich ist, ein Ergebnis einen bestimmten Zweig weiterzuleiten, das Ergebnis in beide Zweige geschickt wird.
5. *Mehrfachteilung* (schwierig). Alle Bäume, die in diesem Kapitel erstellt wurden, sind binäre Entscheidungsbäume. Aber manche Datenmengen würden zu einfacheren Bäumen führen, wenn es einem Knoten möglich wäre, in mehr als zwei Zweige zu verästeln. Wie würden Sie dies repräsentieren? Wie würden Sie den Baum trainieren?

e-bol.net

KAPITEL 8

Aufbauen von Preismodellen

Wir haben schon verschiedene Klassifikatoren untersucht, die meist dann gut funktionieren, wenn es darum geht, eine Kategorie vorherzusagen, in die ein neues Datenelement passt. Aber Bayes-Klassifikatoren, Entscheidungsbäume und Support-Vector-Maschinen (die Sie im nächsten Kapitel kennenlernen werden) sind nicht die besten Algorithmen, um Vorhersagen zu numerischen Daten zu machen, die auf vielen verschiedenen Attributen basieren – wie zum Beispiel Preise. Dieses Kapitel wird sich mit trainierbaren Algorithmen befassen, die numerische Vorhersagen treffen und dabei auf Beispielen aufbauen, die ihnen vorher präsentiert wurden. Dazu können sie sogar Wahrscheinlichkeitsverteilungen für die Vorhersagen angeben, um dem Benutzer bei der Interpretation zu helfen, wie die Vorhersage getroffen wurde.

Wir werden uns anschauen, wie man diese Algorithmen nutzt, um Modelle zur Preisvorhersage zu erstellen. Ökonomen sehen Preise, insbesondere Auktionspreise, als gute Methode an, mithilfe der kollektiven Intelligenz den wahren Wert von etwas zu bestimmen – in einem großen Markt mit vielen Käufern und Verkäufern wird der Preis normalerweise den optimalen Wert für beide an der Transaktion beteiligten Seiten annehmen. Eine Preisvorhersage ist zudem ein guter Test für solche Algorithmen, da es normalerweise viele verschiedene Faktoren gibt, die man beim Bestimmen eines Preises berücksichtigen muss. Wenn man zum Beispiel auf einen Laptop bietet, muss man die Prozessorleistung, das vorhandene RAM, die Größe der Festplatte, die Bildschirmauflösung und andere Faktoren in Betracht ziehen.

Ein wichtiger Teil beim Vorhersagen numerischer Werte ist die Bestimmung der wichtigen Variablen und der entscheidenden Kombinationen. Im Laptop-Beispiel wird es sehr wahrscheinlich viele Variablen geben, die – wenn überhaupt – den Preis nur marginal beeinflussen werden, wie zum Beispiel zusätzliche Hardwareteile oder dazu mitgelieferte Software. Dagegen wird die Bildschirmgröße einen größeren Einfluss auf den abschließenden Preis haben als die Festplattengröße. Sie werden die in Kapitel 5 entwickelten Optimierungstechniken nutzen, um automatisch die besten Gewichtungen für die Variablen zu bestimmen.

e-bol.net

Aufbau eine Beispiel-Datenmenge

Eine interessante Datenmenge zum Testen eines Algorithmus zur Voraussage numerischer Werte sollte ein paar Eigenschaften besitzen, die eine Vorhersage erschweren. Wenn Sie sich Fernseher anschauen, ist die Schlussfolgerung einfach, dass größer auch besser ist. Solche Probleme lassen sich einfacher mit klassischen statistischen Techniken lösen. Aus diesem Grund ist es nützlicher, sich eine Datenmenge anzuschauen, bei der die Preise nicht einfach proportional zur Größe oder Anzahl der Eigenschaften steigen.

In diesem Abschnitt werden Sie eine Datenmenge mit Weinpreisen erstellen, die auf einem einfachen, künstlichen Modell basiert. Die Preise ergeben sich aus der Kombination von Bewertung und Alter des Weins. Das Modell geht davon aus, dass Wein ein bestes Alter besitzt, das für gute Weine älter ist, während es für schlechte Weine fast schon beim Abfüllen vorbei ist. Ein gut gewerteter Wein wird mit einem hohen Preis beginnen und seinen Wert noch steigern, bis sein bestes Alter erreicht ist, während ein schlecht bewerteter Wein günstig beginnt und dann noch billiger wird.

Um dies zu modellieren, erstellen Sie eine neue Datei namens *numpredict.py* und fügen die Funktion `wineprice` hinzu:

```
from random import random,randint
import math

def wineprice(rating,age):
  peak_age=rating-50

  # Preis aufgrund der Bewertung berechnen.
  price=rating/2
  if age>peak_age:
    # Wird nach dem besten Alter in 5 Jahren schlecht.
    price=price*(5-(age-peak_age))
  else:
    # Steigt auf den fünffachen Ursprungswert, wenn das
    # beste Alter erreicht wird.
    price=price*(5*((age+1)/peak_age))
  if price<0: price=0
  return price
```

Sie werden auch eine Funktion brauchen, um eine Datenmenge mit Weinpreisen aufzubauen. Die folgende Funktion erzeugt 300 Flaschen Wein und berechnet ihre Preise über das Modell. Dann fügt sie zufällig bis zu 20% hinzu oder zieht sie ab, um verschiedene Einflussfaktoren zu berücksichtigen, wie zum Beispiel Steuern oder das lokale Preisniveau. Zudem soll die numerische Vorhersage dadurch ein bisschen schwieriger werden. Fügen Sie die Funktion `wineset1` der Datei *numpredict.py* hinzu:

e-bol.net

```
def wineset1( ):
  rows=[]
  for i in range(300):
    # Zufallsalter und Bewertung erzeugen.
    rating=random( )*50+50
    age=random( )*50

    # Referenzpreis erhalten.
    price=wineprice(rating,age)

    # Rauschen addieren.
    price*=(random( )*0.4+0.8)

    # Der Datenmenge hinzufügen.
    rows.append({'input':(rating,age),
                 'result':price})
  return rows
```

Starten Sie eine Python-Session, testen Sie ein paar Weinpreise und bauen Sie eine neue Datenmenge auf:

```
$ python
>>> import numpredict
>>> numpredict.wineprice(95.0,3.0)
21.111111111111114
>>> numpredict.wineprice(95.0,8.0)
47.5
>>> numpredict.wineprice(99.0,1.0)
10.102040816326529
>>> data=numpredict.wineset1( )
>>> data[0]
{'input': (63.602840187200407, 21.574120872184949), 'result': 34.565257353086487}
>>> data[1]
{'input': (74.994980945756794, 48.052051269308649), 'result': 0.0}
```

In der gezeigten Datenmenge ist die zweite Flasche zu alt und daher nicht mehr genießbar, während die erste gerade richtig gealtert ist. Das Zusammenspiel der Variablen lässt dies eine gute Datenmenge sein, mit der Algorithmen getestet werden können.

k-nächste Nachbarn

Der einfachste Ansatz für unser Weinpreisproblem ist der gleiche, den Sie nutzen würden, wenn Sie etwas ohne Computerhilfe bepreisen müssten – ein paar möglichst ähnliche Objekte finden und davon ausgehen, dass die Preise ungefähr gleich sein werden. Wenn eine Reihe von Objekten gefunden wird, die dem fraglichen ähneln, kann der Algorithmus deren Preise mitteln und einen Vorschlag dazu machen, wie der Preis für das Objekt sein sollte. Dieses Vorgehen wird als *k-nächste Nachbarn* (*k-nearest neighbor*, kNN) bezeichnet.

e-bol.net

Anzahl der Nachbarn

Das k in kNN bezieht sich auf die Anzahl der Objekte, die gemittelt werden, um das Ergebnis zu erhalten. Wären die Daten perfekt, könnten Sie k=1 nutzen, also den nächsten Nachbarn heraussuchen und seinen Preis als Antwort verwenden. Aber in der Realität gibt es immer Abweichungen. Im Beispiel haben Sie absichtlich ein »Rauschen« hinzugefügt, um dies zu simulieren (die zufällige Addition oder Subtraktion von bis zu 20%). Vielleicht hat jemand ein echtes Schnäppchen ergattert, oder ein uninformierter Kunde hat deutlich zu viel für den nächsten Nachbarn bezahlt. Aus diesem Grund ist es besser, ein paar Nachbarn auszuwählen und sie zu mitteln, um ein Rauschen herauszurechnen.

Um das Problem der Auswahl von zu wenig Nachbarn sichtbar zu machen, stellen Sie sich ein Problem vor, bei dem es nur eine beschreibende Variable gibt, wie zum Beispiel das Alter. Abbildung 8-1 zeigt einen Graphen mit Preisen (auf der y-Achse) versus Alter (auf der x-Achse). Im Graphen sieht man auch die Linie, die Sie erhalten, wenn Sie nur einen einzelnen nächsten Nachbarn wählen.

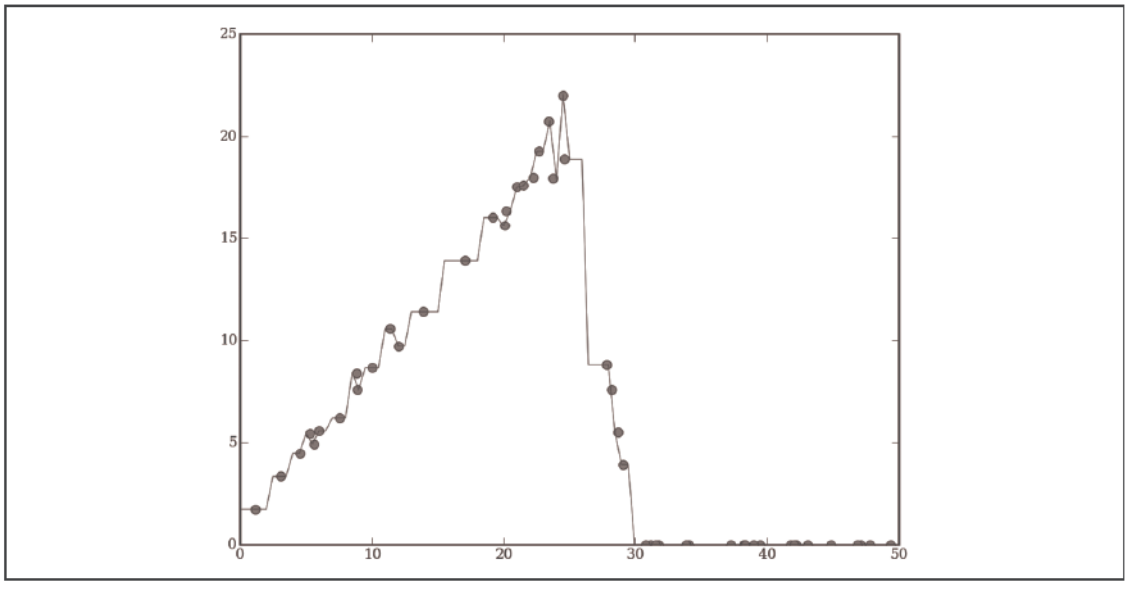

Abbildung 8-1: kNN mit zu wenig verwendeten Nachbarn

Beachten Sie, wie stark die vorhergesagten Preise von zufälligen Änderungen abhängen. Wenn Sie die sprunghafte Linie nutzen würden, um eine Vorhersage zu treffen, würden Sie damit festlegen, dass es einen großen Preissprung gibt zwischen Wein, der 15 Jahre alt ist, und Wein, der 16 Jahre alt ist – wo es sich in Wirklichkeit doch nur um den Unterschied der Preise zwischen zwei bestimmten Flaschen handelt.

Andererseits kann eine Wahl von zu vielen Nachbarn die Genauigkeit verringern, da der Algorithmus dann die Daten von Objekten in Betracht ziehen wird, die dem fraglichen Objekt so gar nicht mehr ähnlich sind. Abbildung 8-2 zeigt die gleiche Datenmenge, wobei die Linie die 20 nächsten Nachbarn mittelt.

e-bol.net

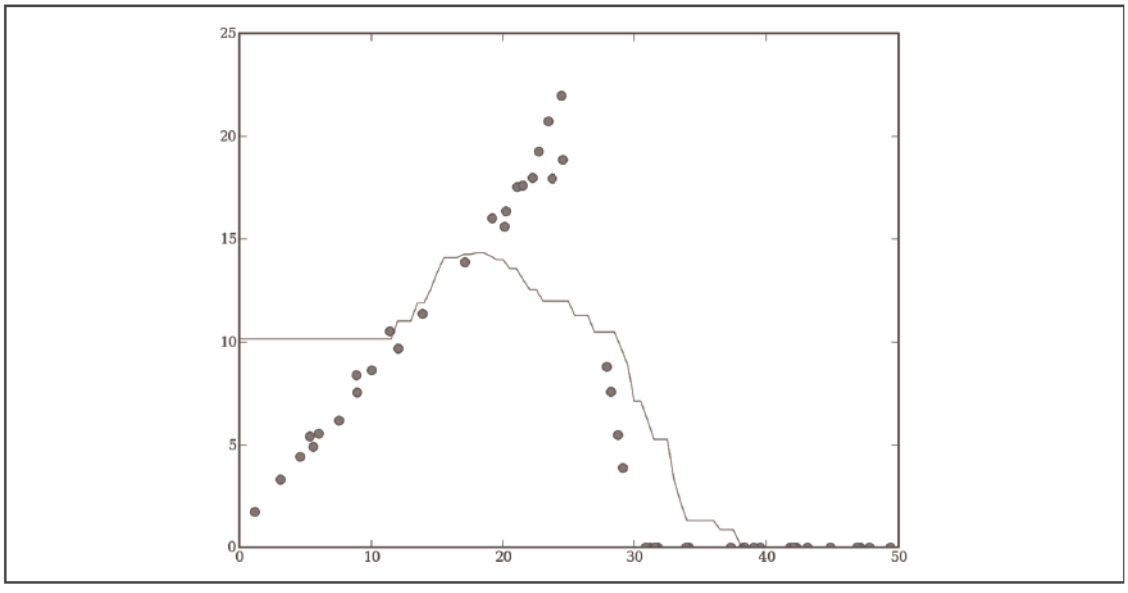

Abbildung 8-2: kNN mit zu vielen Nachbarn

Es ist offensichtlich, dass ein Mitteln von zu vielen Preisen die Weine im Bereich von 25 Jahren deutlich zu billig einstuft. Die richtige Wahl der Anzahl an Nachbarn kann für verschiedene Datenmengen per Hand vorgenommen werden, oder man greift auf eine Optimierung zurück.

Ähnlichkeit definieren

Als Erstes brauchen Sie für den kNN-Algorithmus eine Möglichkeit, die Ähnlichkeit zweier Objekte zu messen. Sie haben in diesem Buch schon ein paar Metriken dafür kennengelernt. Hier werden Sie den euklidischen Abstand nutzen, eine Funktion, die wir schon in mehreren Kapiteln behandelt haben. Fügen Sie die Funktion `euclidian` der Datei *numpredict.py* hinzu:

```
def euclidean(v1,v2):
  d=0.0
  for i in range(len(v1)):
    d+=(v1[i]-v2[i])**2
  return math.sqrt(d)
```

Testen Sie die Funktion in Ihrer Python-Session mit ein paar Elementen aus Ihrer Datenmenge aus:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> data[0]['input']
(82.720398223643514, 49.21295829683897)
>>> data[1]['input']
(98.942698715228076, 25.702723509372749)
>>> numpredict.euclidean(data[0]['input'],data[1]['input'])
28.56386131112269
```

e-bol.net

Sie werden feststellen, dass diese Funktion Alter und Bewertung beim Berechnen des Abstands gleich bewertet, obwohl in so gut wie allen realen Problemen einige Variablen einen größeren Einfluss auf den abschließenden Preis haben als andere. Dies ist eine bekannte Schwäche des kNN, aber Sie werden in diesem Kapitel noch Möglichkeiten kennenlernen, dies zu korrigieren.

Code für k-nächste Nachbarn

kNN lässt sich recht einfach implementieren. Er ist rechenintensiv, hat aber den Vorteil, dass beim Hinzufügen von neuen Daten kein neues Training notwendig ist. Fügen Sie die Funktion `getdistances` *numpredict.py* hinzu, um die Abstände zwischen einem gegebenen Element und jedem anderen Element in der ursprünglichen Datenmenge zu bestimmen:

```
def getdistances(data,vec1):
  distancelist=[]
  for i in range(len(data)):
    vec2=data[i]['input']
    distancelist.append((euclidean(vec1,vec2),i))
  distancelist.sort( )
  return distancelist
```

Diese Funktion ruft die Funktion `distance` für den gegebenen Vektor und jeden anderen Vektor in der Datenmenge auf und trägt das Ergebnis in eine große Liste ein. Diese wird dann sortiert, sodass das naheste Objekt am Anfang steht.

Die kNN-Funktion nutzt die Abstandsliste und mittelt die obersten `k` Ergebnisse. Fügen Sie `knnestimate` der Datei *numpredict.py* hinzu:

```
def knnestimate(data,vec1,k=3):
  # Sortierte Abstände erhalten.
  dlist=getdistances(data,vec1)
  avg=0.0

  # Durchschnitt der obersten k Ergebnisse nehmen.
  for i in range(k):
    idx=dlist[i][1]
    avg+=data[idx]['result']
  avg=avg/k
  return avg
```

Sie können nun eine Preisabschätzung für ein neues Objekt erhalten:

```
>>> reload(numpredict)
>>> numpredict.knnestimate(data,(95.0,3.0))
29.176138546872018
>>> numpredict.knnestimate(data,(99.0,3.0))
22.356856188108672
>>> numpredict.knnestimate(data,(99.0,5.0))
37.610888778473793
>>> numpredict.wineprice(99.0,5.0)  # echten Preis berechnen
```

e-bol.net

```
30.306122448979593
>>> numpredict.knnestimate(data,(99.0,5.0),k=1) # weniger Nachbarn
38.078819347238685
```

Probieren Sie verschiedene Parameter und unterschiedliche Werte für k aus, um zu sehen, wie die Ergebnisse beeinflusst werden.

Gewichtete Nachbarn

Eine Möglichkeit, die Tatsache zu kompensieren, dass der Algorithmus eventuell zu weit entfernte Nachbarn nutzt, ist ein Gewichten in Abhängigkeit vom Abstand. Das ähnelt der in Kapitel 2 genutzten Methode, bei der die Vorlieben der Personen danach gewichtet wurden, wie ähnlich sie den Vorlieben der fraglichen Person waren, die eine Empfehlung suchte.

Je ähnlicher die Elemente sind, desto kleiner wird der Abstand zwischen ihnen sein, sodass Sie einen Weg brauchen, Abstände in Gewichtungen umzurechnen. Es gibt ein paar Möglichkeiten dafür, von denen jede ihre Vor- und Nachteile hat. Dieser Abschnitt wird drei Funktionen vorstellen, die Sie nutzen können.

Inverse Funktion

Die Funktion, die Sie in Kapitel 4 zum Umwandeln von Abständen in Gewichtungen genutzt haben, war eine inverse Funktion. Abbildung 8-3 zeigt, wie diese aussieht, wenn Sie die Gewichtung auf einer Achse auftragen und den Abstand auf der anderen.

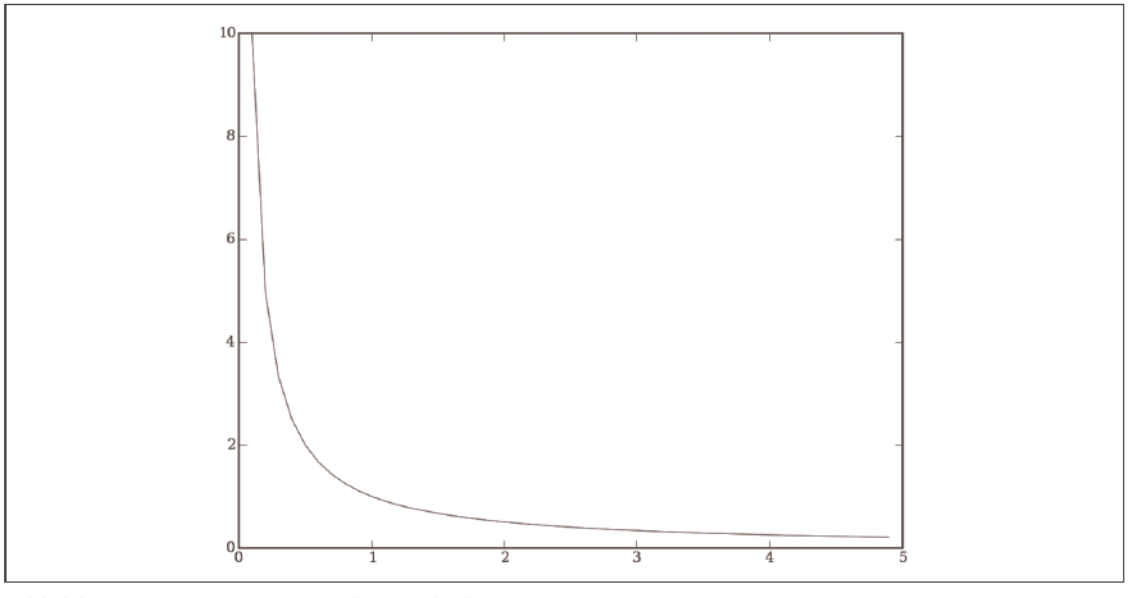

Abbildung 8-3: Inverse Gewichtungsfunktion

e-bol.net

Die einfachste Form dieser Funktion liefert 1 geteilt durch den Abstand zurück. Aber in manchen Fällen sind Objekte genau gleich oder liegen sehr nahe beieinander, was zu einer sehr großen oder sogar unendlichen Gewichtung führt. Aus diesem Grund ist es notwendig, dem Abstand einen kleinen Wert hinzuzufügen, bevor man ihn invertiert.

Fügen Sie die Funktion `inverseweight` *numpredict.py* hinzu:

```
def inverseweight(dist,num=1.0,const=0.1):
  return num/(dist+const)
```

Diese Funktion ist schnell und lässt sich einfach implementieren. Dazu können mit verschiedenen Werten für `num` experimentieren, um herauszufinden, welche Werte gute Ergebnisse liefern. Der entscheidende mögliche Nachteil ist, dass für Objekte sehr große Gewichtungen angewendet werden, die sehr nahe liegen, die Gewichtung aber mit zunehmendem Abstand schnell sehr klein wird. Das kann erwünscht sein, aber in manchen Fällen wird der Algorithmus dadurch rauschempfindlicher.

Subtraktionsfunktion

Eine zweite Option ist die *Subtraktionsfunktion*, deren Graph in Abbildung 8-4 zu sehen ist.

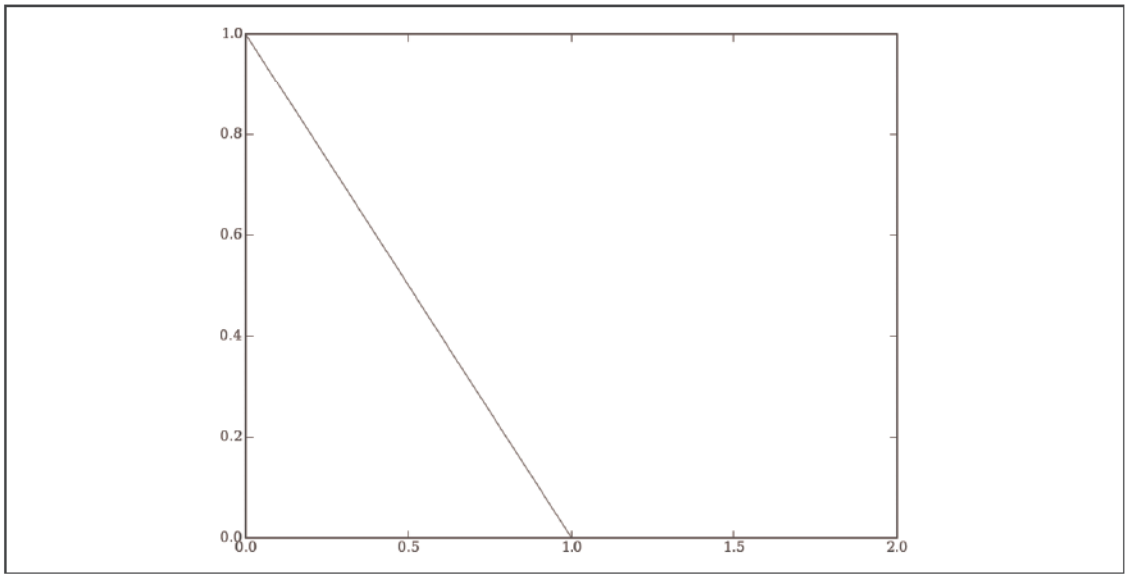

Abbildung 8-4: Subtraktions-Gewichtungsfunktion

Dies ist eine einfache Funktion, die den Abstand von einer Konstanten abzieht. Die Gewichtung ist das Ergebnis dieser Subtraktion, wenn das Ergebnis größer als null ist, ansonsten wird null zurückgegeben. Fügen Sie die Funktion `subtractweight` der Datei *numpredict.py* hinzu:

```
def subtractweight(dist,const=1.0):
  if dist>const:
```

e-bol.net

```
    return 0
  else:
    return const-dist
```

Diese Funktion hat nicht das Problem der Übergewichtung von nahe beieinander liegenden Objekten, aber sie hat auch ihre eigenen Nachteile. Da die Gewichtung irgendwann auf 0 fällt, ist es möglich, das nichts nahe genug liegt, um als Nachbar gewertet zu werden, sodass der Algorithmus für manche Objekte gar keine Vorhersage treffen kann.

Gauß-Funktion

Die letzte vorgestellte Funktion ist eine *Gauß-Funktion*, auch als *Glockenkurve* bekannt. Sie ist ein wenig komplexer als die anderen hier behandelten Funktionen, aber wie Sie sehen werden, hat sie nicht so viele Einschränkungen. Die Gauß-Funktion wird in Abbildung 8-5 gezeigt.

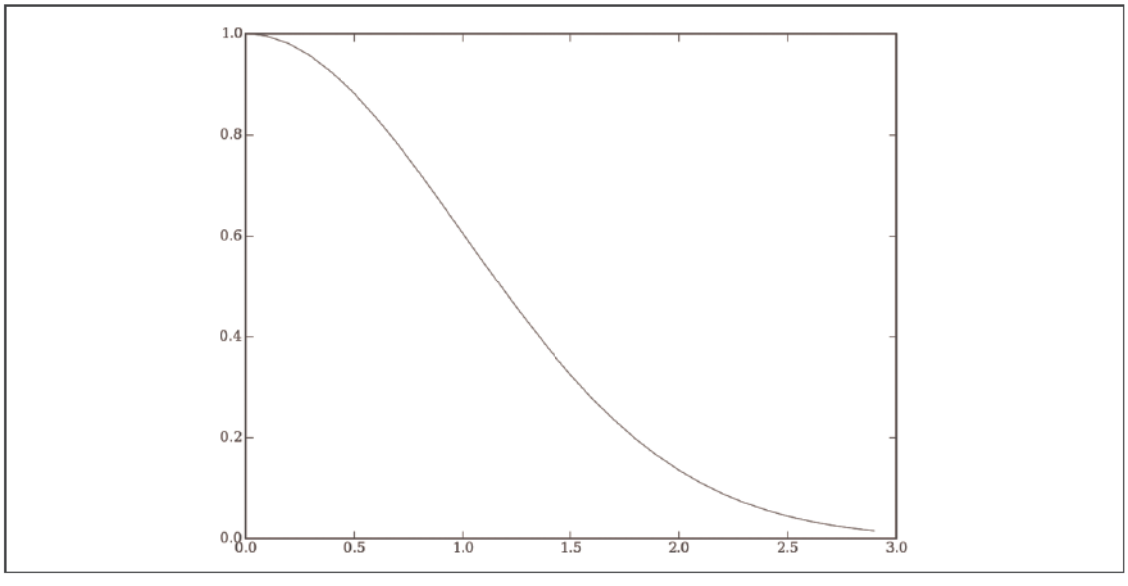

Abbildung 8-5: Gauß-Gewichtungsfunktion

Die Gewichtung in dieser Funktion ist 1, wenn der Abstand 0 ist, dann wird sie bei zunehmendem Abstand kleiner. Aber anders als die Subtraktionsfunktion fällt die Gewichtung niemals vollständig auf 0, sodass es immer möglich sein wird, eine Vorhersage zu treffen. Der Code für diese Funktion ist komplexer und wird auch nicht so schnell zu berechnen sein wie die beiden anderen Funktionen.

Fügen Sie `gaussian` der Datei *numpredict.py* hinzu:

```
def gaussian(dist,sigma=10.0):
  return math.e**(-dist**2/(2*sigma**2))
```

Sie werden nun die verschiedenen Funktionen für ein paar der Elemente mit verschiedenen Parametern testen und die Unterschiede analysieren:

e-bol.net

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.subtractweight(0.1)
0.9
>>> numpredict.inverseweight(0.1)
5.0
>>> numpredict.gaussian(0.1)
0.99501247919268232
>>> numpredict.gaussian(1.0)
0.60653065971263342
>>> numpredict.subtractweight(1)
0.0
>>> numpredict.inverseweight(1)
0.90909090909090906
>>> numpredict.gaussian(3.0)
0.01110899653824231
```

Sie können feststellen, dass alle Funktionen ihre höchsten Werte bei 0,0 haben und dann auf unterschiedliche Art und Weise mit zunehmendem Abstand fallen.

Gewichtete kNN

Der Code für eine gewichtete kNN-Funktion arbeitet wie die normale kNN-Funktion – es werden die sortierten Abstände ermittelt und die `k` nahesten Elemente genutzt. Der Unterschied liegt darin, dass statt einer einfachen Mittelung nun ein *gewichtetes Mittel* berechnet wird. Das gewichtete Mittel wird errechnet, indem die Gewichtung jedes Objekts mit seinem Wert multipliziert wird, bevor man ihn addiert. Nach Ermitteln der Summe wird sie durch die Summe aller Gewichtungen geteilt.

Fügen Sie `weightedknn` der Datei *numpredict.py* hinzu:

```
def weightedknn(data,vec1,k=5,weightf=gaussian):
  # Abstände ermitteln.
  dlist=getdistances(data,vec1)
  avg=0.0
  totalweight=0.0

  # Gewichteter Durchschnitt.
  for i in range(k):
    dist=dlist[i][0]
    idx=dlist[i][1]
    weight=weightf(dist)
    avg+=weight*data[idx]['result']
    totalweight+=weight
  avg=avg/totalweight
  return avg
```

Die Funktion läuft über die `k` nächsten Nachbarn und übergibt jeden Abstand an eine der Gewichtungsfunktionen, die Sie weiter oben definiert haben. Die Variable `avg` wird berechnet, indem diese Gewichtungen mit dem Wert des Nachbarn multi-

e-bol.net

pliziert werden. Die Variable totalweight ist die Summe der Gewichtungen. Schließlich wird avg durch totalweight dividiert.

Sie können diese Funktion in Ihrer Python-Session austesten und ihre Performance mit der der normalen kNN-Funktion vergleichen:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.weightedknn(data,(99.0,5.0))
32.640981119354301
```

In diesem Beispiel zeigt sich, dass sich weightedknn der korrekten Antwort besser annähert als knnestimate. Aber das gilt nur für ein paar Fälle. Ein ordentlicher Test würde viele Objekte aus der Datenmenge mit einbeziehen, was Sie dann auch nutzen könnten, um den besten Algorithmus und die besten Parameter zu ermitteln. Als Nächstes werden Sie Möglichkeiten kennenlernen, solch einen Test durchzuführen.

Kreuzvalidierung

Die Kreuzvalidierung ist der Name einer Reihe von Techniken, die Daten in *Trainings-* und *Testsets* aufteilen. Das Trainingsset wird dem Algorithmus zusammen mit den korrekten Antworten übergeben (in diesem Fall den Preisen) und genutzt, um damit Voraussagen zu treffen. Der Algorithmus wird dann verwendet, um Vorhersagen für jedes Element im Testset zu erstellen. Die gelieferten Antworten werden mit den korrekten Antworten verglichen, und eine Gesamtbewertung für die Qualität des Algorithmus wird berechnet.

Normalerweise wird diese Prozedur viele Male durchgeführt, wobei die Daten jedes Mal anders unterteilt werden. Typischerweise ist das Testset nur klein, vielleicht 5 % der Daten, während die restlichen 95 % das Trainingsset bilden. Dazu erstellen Sie nun eine Funktion namens dividedata in *numpredict.py*, die die Datenmenge in zwei kleinere Mengen unterteilt, wobei Sie ein Verhältnis zwischen den Größen der beiden Mengen angeben können:

```
def dividedata(data,test=0.05):
  trainset=[]
  testset=[]
  for row in data:
    if random( )<test:
      testset.append(row)
    else:
      trainset.append(row)
  return trainset,testset
```

Der nächste Schritt ist, den Algorithmus zu testen, indem Sie ihm ein Trainingsset übergeben und ihn dann mit jedem Element des Testsets aufrufen. Die Funktion berechnet die Differenzen und kombiniert sie, um eine Gesamtbewertung zu ermit-

e-bol.net

teln, die angibt, wie groß die Abweichungen insgesamt sind. Das geschieht normalerweise, indem man die Quadrate aller Differenzen addiert.

Fügen Sie eine neue Funktion `testalgorithm` der Datei *numpredict.py* hinzu:

```
def testalgorithm(algf,trainset,testset):
  error=0.0
  for row in testset:
    guess=algf(trainset,row['input'])
    error+=(row['result']-guess)**2
  return error/len(testset)
```

`testalgorithm` erwartet einen Algorithmus `algf`, der eine Datenmenge und eine Abfrage als Parameter hat. Dann läuft die Funktion über jede Zeile in der Testmenge und berechnet die beste Schätzung, indem sie `algf` aufruft. Anschließend werden die Schätzungen von den echten Ergebnissen abgezogen.

Das Quadrieren der Werte wird üblicherweise immer vorgenommen, da dadurch große Differenzen noch stärker ins Gewicht fallen. Das bedeutet, dass ein Algorithmus, der die meiste Zeit sehr gut ist, aber ab und zu richtig danebenhaut, deutlich schlechter gewertet wird als ein Algorithmus, der immer halbwegs gut ist. Das ist meist das gewünschte Verhalten, aber es gibt auch Situationen, in denen gelegentlich ein großer Fehler akzeptabel ist, wenn ansonsten die Genauigkeit sehr hoch ist. In solchen Fällen können Sie die Funktion so anpassen, dass sie nur die absoluten Werte der Differenzen addiert.

Im letzten Schritt wird nun eine Funktion erstellt, die die Daten mehrfach unterschiedlich unterteilt und jedes Mal `testalgorithm` aufruft, wobei die Ergebnisse aufsummiert werden, um ein Gesamtergebnis zu erhalten. Fügen Sie `crossvalidate` *numpredict.py* hinzu:

```
def crossvalidate(algf,data,trials=100,test=0.05):
  error=0.0
  for i in range(trials):
    trainset,testset=dividedata(data,test)
    error+=testalgorithm(algf,trainset,testset)
  return error/trials
```

Der Code, den Sie bisher gebaut haben, bietet nun viele Vergleichsmöglichkeiten an. Sie können zum Beispiel `knnestimate` mit verschiedenen Werten von `k` testen.

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.crossvalidate(numpredict.knnestimate,data)
254.06864176819553
>>> def knn3(d,v): return numpredict.knnestimate(d,v,k=3)
...
>>> numpredict.crossvalidate(knn3,data)
166.97339783733005
>>> def knn1(d,v): return numpredict.knnestimate(d,v,k=1)
...
>>> numpredict.crossvalidate(knn1,data)
209.54500183486215
```

e-bol.net

Wie erwartet, führt die Verwendung von zu wenigen oder zu vielen Nachbarn zu schlechten Ergebnissen. In diesem Beispiel ist ein Wert von 3 Nachbarn besser als 1 oder 5. Sie können auch die verschiedenen Gewichtungsfunktionen vergleichen, die Sie für die gewichtete kNN definiert haben, um herauszufinden, welche zu den besten Ergebnissen führt:

```
>>> numpredict.crossvalidate(numpredict.weightedknn,data)
200.34187674254176
>>> def knninverse(d,v):
...   return numpredict.weightedknn(d,v,\\
      weightf=numpredict.inverseweight)
>>> numpredict.crossvalidate(knninverse,data)
148.85947702660616
```

Wenn die Parameter gut gewählt sind, scheinen gewichtete kNN bessere Ergebnisse für diese Datenmenge zu liefern. Die Wahl der korrekten Parameter mag zeitaufwendig sein, aber Sie müssen sie für ein bestimmtes Trainingsset nur einmal durchführen und bei Bedarf aktualisieren, wenn das Trainingsset wächst. Im Abschnitt »Optimieren der Skalierung« weiter unten in diesem Kapitel werden Sie Wege kennenlernen, einige der Parameter automatisch zu bestimmen.

Heterogene Variablen

Die Datenmenge, die Sie zu Beginn dieses Kapitels erstellt haben, war darauf ausgelegt, sehr einfach zu sein – insbesondere sind alle Variablen, die zur Vorhersage des Preises genutzt wurden, halbwegs vergleichbar und haben alle großen Einfluss auf das Endergebnis.

Da alle Variablen im gleichen Bereich liegen, kann man die Abstände sinnvoll mit allen gleichzeitig berechnen. Stellen Sie sich aber einmal vor, Sie würden eine neue Variable einführen, die den Preis beeinflusst, wie zum Beispiel die Größe der Flasche in Millilitern. Anders als die Variablen, die Sie bisher genutzt haben und die alle zwischen 0 und 100 liegen, kann diese bis 1500 gehen. Schauen Sie sich Abbildung 8-6 an, um zu sehen, wie dies die Berechnungen zum nächsten Nachbarn oder zur Gewichtung des Abstands beeinflussen würde.

Ganz offensichtlich hat diese neue Variable einen deutlich größeren Einfluss auf die berechneten Abstände als die ursprünglichen Variablen – sie wird jede Abstandsberechnung beherrschen, was dazu führt, dass die anderen Variablen nicht mehr in Betracht gezogen werden.

Ein anderes Problem ist die Berücksichtigung komplett irrelevanter Variablen. Wenn die Datenmenge auch die Gangnummer enthalten würde, in der Sie den Wein gefunden haben, würde diese Variable die Abstandsberechnung ebenfalls beeinflussen. Zwei ansonsten vollständig identische Objekte, die nur in verschiedenen Gängen gelagert wurden, würden als weit auseinanderliegend bewertet werden, was die Fähigkeit des Algorithmus zu genauen Voraussagen stark behindern würde.

e-bol.net

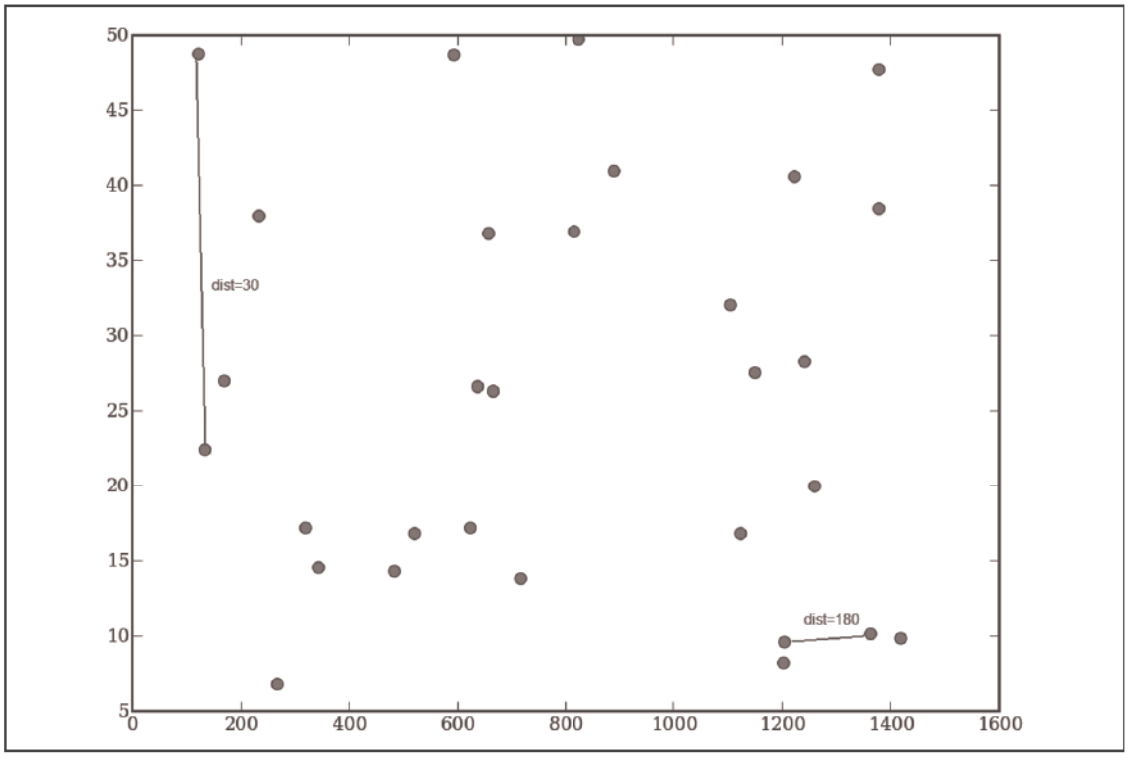

Abbildung 8-6: Heterogene Variablen führen zu Abstandsproblemen

Der Datenmenge etwas hinzufügen

Um diese Effekte zu simulieren, werden Sie Ihrer Datenmenge ein paar neue Variablen hinzufügen. Sie können den Code aus `wineset1` kopieren, um eine neue Funktion namens `wineset2` zu erstellen und die fett gedruckten Zeilen hinzuzufügen:

```
def wineset2( ):
  rows=[]
  for i in range(300):
    rating=random( )*50+50
    age=random( )*50
    aisle=float(randint(1,20))
    bottlesize=[375.0,750.0,1500.0,3000.0][randint(0,3)]
    price=wineprice(rating,age)
    price*=(bottlesize/750)
    price*=(random( )*0.9+0.2)
    rows.append({'input':(rating,age,aisle,bottlesize),
                 'result':price})
  return rows
```

Jetzt können Sie neue Datenmengen mit Gängen und Flaschengrößen erstellen:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> data=numpredict.wineset2( )
```

e-bol.net

Um zu sehen, wie die Vorhersage durch die kNN dadurch beeinflusst wird, testen Sie die neuen Daten mit den besten Parametern, die Sie weiter oben schon gefunden haben:

```
>>> numpredict.crossvalidate(knn3,data)
1427.3377833596137
>>> numpredict.crossvalidate(numpredict.weightedknn,data)
1195.0421231227463
```

Obwohl die Datenmenge nun sogar mehr Informationen und weniger Rauschen enthält als zuvor (was theoretisch zu besseren Voraussagen führen sollte), sind die von `crossvalidate` zurückgegebenen Werte tatsächlich deutlich schlechter geworden. Das liegt daran, dass die Algorithmen noch nicht wissen, wie die Variablen jeweils unterschiedlich zu behandeln sind.

Skalieren der Dimensionen

Wir brauchen hier nicht die Möglichkeit, den Abstand auf den eigentlichen Werten basieren zu lassen, sondern wir müssen die Werte so normalisieren, dass man sie alle im selben Raum behandeln kann. Es wäre auch hilfreich, einen Weg zu finden, überflüssige Variablen herauszuschmeißen oder zumindest ihren Einfluss auf die Berechnungen zu reduzieren. Ein Weg, beides zu erreichen, ist ein Umskalieren der Dimensionen, bevor die Berechnungen durchgeführt werden.

Die einfachste Form der Umskalierung ist ein Multiplizieren der Werte in jeder Dimension mit einer Konstanten (für diese Dimension). Ein Beispiel für die Umskalierung ist in Abbildung 8-7 zu sehen.

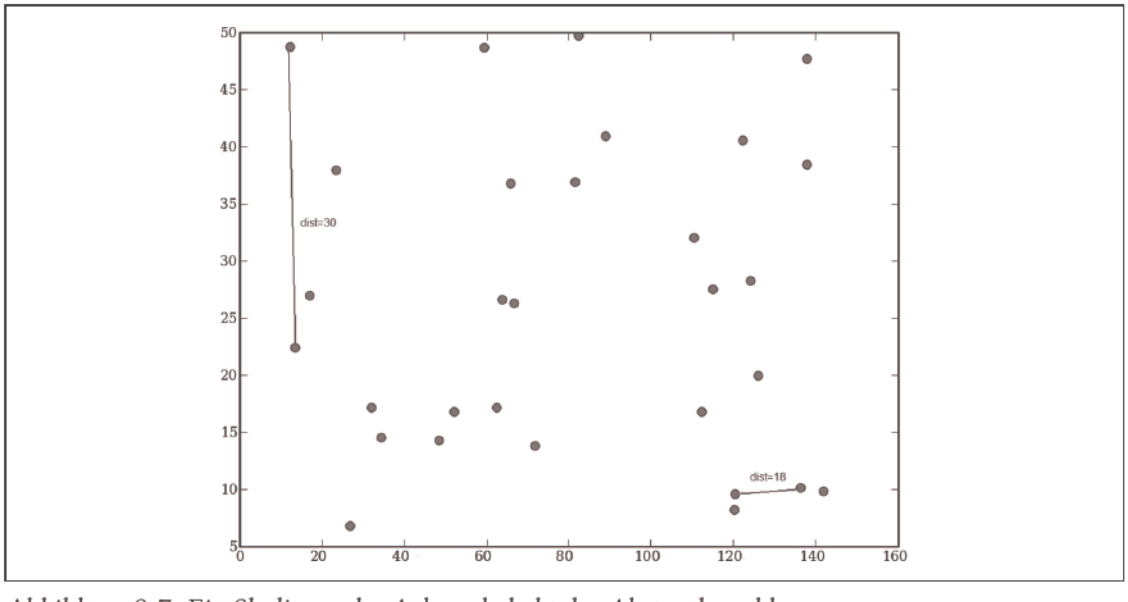

Abbildung 8-7: Ein Skalieren der Achsen behebt das Abstandsproblem

e-bol.net

Sie können sehen, dass die Achse für die Flaschengröße um den Faktor 10 herunterskaliert wurde und sich daher logischerweise die nächsten Nachbarn für viele der Elemente geändert haben. Das löst das Problem, dass manche Variablen größer sind als andere, aber wie sieht es mit den unwichtigen Variablen aus? Was würde passieren, wenn man den Wert jedes Elements in einer Dimension mit 0 multipliziert, wie dies in Abbildung 8-8 zu sehen ist?

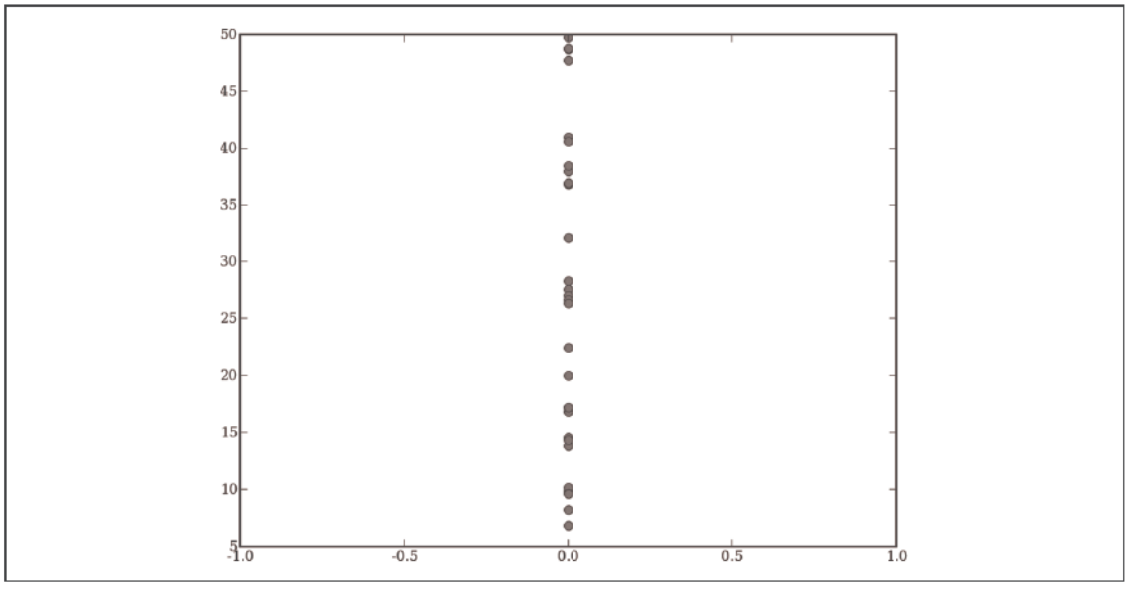

Abbildung 8-8: Unwichtige Achsen werden auf 0 skaliert

In der Gang-Dimension befindet sich nun alles an der gleichen Stelle, daher hängt der Abstand zwischen den Elementen nun vollständig von der Position in der Alters-Dimension ab. Das bedeutet, dass der Gang in der Berechnung der nächsten Nachbarn komplett unwichtig geworden ist und aus den Vorhersagen gestrichen wurde. Sind alle unwichtigen Variablen erst mal auf 0 reduziert, werden die Algorithmen weitaus genauer werden.

Die Funktion `rescale` erwartet eine Liste von Elementen und einen Parameter namens `scale`, der aus einer Liste mit Gleitkommazahlen besteht. Sie liefert eine neue Datenmenge zurück, in der alle Werte mit den Werten aus `scale` multipliziert wurden. Fügen Sie `rescale` der Datei *numpredict.py* hinzu:

```
def rescale(data,scale):
  scaleddata=[]
  for row in data:
    scaled=[scale[i]*row['input'][i] for i in range(len(scale))]
    scaleddata.append({'input':scaled,'result':row['result']})
  return scaleddata
```

Sie können dies austesten, indem Sie eine Datenmenge mit ein paar sinnvoll gewählten Parametern umskalieren und sehen, ob die Vorhersagen wieder besser werden:

e-bol.net

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> sdata=numpredict.rescale(data,[10,10,0,0.5])
>>> numpredict.crossvalidate(knn3,sdata)
660.9964024835578
>>> numpredict.crossvalidate(numpredict.weightedknn,sdata)
852.32254222973802
```

Die Ergebnisse sind für diese paar Beispiele ziemlich gut – wahrscheinlich besser als vorher. Ändern Sie die Skalierungsparameter probehalber einmal und schauen Sie, ob Sie die Ergebnisse noch weiter verbessern können.

Optimieren der Skalierung

In diesem Fall ist es nicht schwierig, gute Parameter für die Skalierung zu finden, da Sie im Voraus wissen, welche Variablen wichtig sind. Aber meist werden Sie mit Datenmengen arbeiten, die Sie nicht selbst aufgebaut haben und bei denen Sie nicht notwendigerweise wissen, welche Variablen unwichtig sind und welche einen großen Einfluss haben.

Theoretisch könnten Sie viele Werte in unterschiedlichsten Kombinationen ausprobieren, bis sie eine finden, die gut genug funktioniert, aber es kann Hunderte von Variablen geben, die berücksichtigt werden müssten – das kann nervig werden. Falls Sie sich durch Kapitel 5 gearbeitet haben, wissen Sie glücklicherweise schon, wie man automatisch eine gute Lösung findet, wenn viele Eingabevariablen zu berücksichtigen sind – per *Optimierung*.

Sie erinnern sich vielleicht, dass Sie bei der Optimierung nur eine Domäne festlegen müssen, die die Anzahl der Variablen, einen Bereich und eine Zielfunktion definiert. Die Funktion `crossvalidate` liefert einen höheren Wert für schlechtere Lösungen zurück, daher ist sie schon als Zielfunktion nutzbar. Sie müssen sie nur noch mit einer anderen Funktion umhüllen, sodass sie eine Liste mit Werten als Parameter erwartet, die Daten umskaliert und den Kreuzvalidierungsfehler berechnet. Fügen Sie `createcostfunction` der Datei *numpredict.py* hinzu:

```
def createcostfunction(algf,data):
  def costf(scale):
    sdata=rescale(data,scale)
    return crossvalidate(algf,sdata,trials=10)
  return costf
```

Die Domäne ist der Bereich der Gewichtungen für jede Dimension. In diesem Fall ist der kleinstmögliche Wert 0, da negative Werte nur ein gespiegeltes Bild der Daten liefern, was bei der Abstandsberechnung nichts ändert. Theoretisch können die Gewichtungen so hoch sein, wie Sie wollen, aber aus praktischen Gründen wollen wir sie hier auf 20 begrenzen. Fügen Sie *numpredict.py* diese Zeile hinzu:

```
weightdomain=[(0,20)]*4
```

e-bol.net

Sie haben nun alles, was Sie brauchen, um die Gewichtungen automatisch zu optimieren. Stellen Sie sicher, dass sich *optimization.py*, die in Kapitel 5 erstellte Datei, in Ihrem aktuellen Verzeichnis befindet, und probieren Sie eine Optimierung per Abkühlung in Ihrer Python-Session:

```
>>> import optimization
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.pyc'>
>>> costf=numpredict.createcostfunction(numpredict.knnestimate,data)
>>> optimization.annealingoptimize(numpredict.weightdomain,costf,step=2)
[11,18,0,6]
```

Perfekt! Der Algorithmus stellt nicht nur fest, dass die Gangnummer eine unnütze Variable ist, und reduziert ihre Skalierung auf 0 – er bemerkt auch, dass die Flaschengröße im Vergleich zum Einfluss unverhältnismäßig groß ist, und erhöht die Skalierung der beiden anderen Variablen entsprechend.

Sie können auch die langsamere, aber häufig genauere Funktion `geneticoptimize` austesten und probieren, ob ähnliche Ergebnisse zurückgegeben werden:

```
>>> optimization.geneticoptimize(numpredict.weightdomain,costf,popsize=5,\\
    lrate=1,maxv=4,iters=20)
[20,18,0,12]
```

Ein Vorteil des Optimierens der Variablenskalen auf diese Art und Weise ist, dass Sie direkt sehen, welche Variablen wie wichtig sind. In manchen Fällen lassen sich bestimmte Daten nur sehr schwer oder mit hohen Kosten ermitteln. Wenn Sie dann feststellen, dass sie keinen großen Einfluss auf die Vorhersagen haben, sollten Sie diese Kosten vermeiden. In anderen Fällen kann das Wissen über die Wichtigkeit bestimmter Variablen – insbesondere beim Bestimmen des Preises – einen Einfluss auf Ihre Werbung haben oder zeigen, wie sich Produkte anders gestalten lassen, um höhere Preise zu erzielen.

Ungleiche Verteilungen

Bisher sind wir davon ausgegangen, dass Sie eine recht gute Schätzung des Endpreises erhalten, wenn Sie den Durchschnitt oder den gewichteten Durchschnitt der Daten nutzen. Das wird in vielen Fällen korrekt sein, aber in manchen Situationen kann es nicht erfasste Variablen geben, die einen großen Einfluss auf das Ergebnis haben. Stellen Sie sich im Wein-Beispiel vor, dass Käufer in zwei verschiedene Gruppen eingeteilt werden: Personen, die im Weinkontor kaufen, und Personen, die den Wein im Supermarkt gekauft haben und 50 % Rabatt bekamen. Leider findet sich diese Information nicht in der Datenmenge.

Die Funktion `createhiddendataset` erzeugt eine Datenmenge, die diese Eigenschaften simuliert. Sie verwirft ein paar der verkomplizierenden Variablen und konzentriert sich auf die ursprünglichen. Fügen Sie diese Funktion der Datei *numericalpredictor.py* hinzu:

e-bol.net

```
def wineset3( ):
  rows=wineset1( )
  for row in rows:
    if random( )<0.5:
      # Wein wurde im Supermarkt gekauft.
      row['result']*=0.6
  return rows
```

Stellen Sie sich vor, was passieren würde, wenn Sie mit dem kNN- oder gewichteten kNN-Algorithmus nach einer Abschätzung des Preises für ein anderes Objekt fragten. Da die Datenmenge keine Informationen darüber enthält, ob der Käufer im Weinkontor oder im Supermarkt eingekauft hat, wird der Algorithmus dies auch nicht berücksichtigen können und damit die nächsten Nachbarn liefern, egal wo gekauft wurde. Das Ergebnis ist, dass der Durchschnitt aus Elementen beider Gruppen genommen wird, was möglicherweise in einem Rabatt von 25 % endet. Sie können dies überprüfen, indem Sie es in Ihrer Python-Session ausprobieren:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> data=numpredict.wineset3( )
>>> numpredict.wineprice(99.0,20.0)
106.07142857142857
>>> numpredict.weightedknn(data,[99.0,20.0])
83.475441632209339
>>> numpredict.crossvalidate(numpredict.weightedknn,data)
599.51654107008562
```

Das ist vielleicht nicht so schlecht, wenn Sie nur einen einzelnen Wert haben wollen, aber es spiegelt nicht das wider, was jemand nachher für ein Objekt bezahlen wird. Um die Durchschnittswerte zu vermeiden, müssen Sie einen Weg finden, sich die Daten genauer anzuschauen.

Abschätzen der Wahrscheinlichkeitsdichte

Anstatt den gewichteten Durchschnitt der Nachbarn zu nutzen und eine einzelne Preisschätzung vorzunehmen, kann es in diesem Fall interessant sein, die Wahrscheinlichkeit dafür zu kennen, dass ein Element in einen bestimmten Preisbereich fällt. Hat man beispielsweise als Eingabewerte 99 % und 20 Jahre, hätten Sie sicher gern eine Funktion, die Ihnen mitteilt, dass es eine 50-prozentige Chance dafür gibt, dass der Preis zwischen 40 Dollar und 80 Dollar liegt, während die anderen 50 % sagen, dass der Preis zwischen 80 Dollar und 100 Dollar liegt.

Dazu benötigen Sie eine Funktion, die einen Wert zwischen 0 und 1 zurückgibt, der die Wahrscheinlichkeit repräsentiert. Die Funktion berechnet zunächst die Gewichtungen der Nachbarn innerhalb dieses Bereichs und ermittelt dann die aller Nachbarn. Die Wahrscheinlichkeit ist die Summe dieser Nachbargewichtungen innerhalb des Bereichs, dividiert durch die Summe aller Gewichtungen. Erstellen Sie eine

e-bol.net

neue Funktion namens probguess in *numpredict.py*, um diese Berechnungen durchzuführen:

```
def probguess(data,vec1,low,high,k=5,weightf=gaussian):
  dlist=getdistances(data,vec1)
  nweight=0.0
  tweight=0.0

  for i in range(k):
    dist=dlist[i][0]
    idx=dlist[i][1]
    weight=weightf(dist)
    v=data[idx]['result']

    # Liegt dieser Punkt im Bereich?
    if v>=low and v<=high:
      nweight+=weight
    tweight+=weight
  if tweight==0: return 0

  # Die Wahrscheinlichkeit sind die Gewichtungen im Bereich,
  # geteilt durch alle Gewichtungen.
  return nweight/tweight
```

Wie bei kNN sortiert diese Funktion die Daten nach dem Abstand von vec1 und bestimmt die Gewichtungen der nächsten Nachbarn. Sie summiert die Gewichtungen aller Nachbarn, um tweight zu erhalten. Zudem berücksichtigt sie, ob der Preis jedes Nachbarn im Bereich liegt (zwischen low und high). Wenn das der Fall ist, addiert er die Gewichtung zu nweight. Die Wahrscheinlichkeit, dass der Preis für vec1 zwischen low und high liegt, ist nweight geteilt durch tweight.

Nun probieren Sie diese Funktion mit Ihre Datenmenge aus:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.probguess(data,[99,20],40,80)
0.62305988451497296
>>> numpredict.probguess(data,[99,20],80,120)
0.37694011548502687
>>> numpredict.probguess(data,[99,20],120,1000)
0.0
>>> numpredict.probguess(data,[99,20],30,120)
1.0
```

Die Funktion liefert gute Ergebnisse. Die Bereiche, die außerhalb der tatsächlichen Preise liegen, haben Wahrscheinlichkeiten von 0, und diejenigen, die den gesamten Bereich aller möglichen Preise abdecken, haben als Wert 1. Indem Sie kleinere Bereiche festlegen, können Sie die tatsächliche Verteilung bestimmen, wo sich die Objekte »ballen«. Aber dafür ist erforderlich, dass Sie die Bereiche selbst abschätzen und mit ihnen arbeiten, bis Ihnen klar ist, wie die Struktur der Daten aussieht. Im nächsten Abschnitt werden Sie Wege kennenlernen, um solch einen Überblick über Wahrscheinlichkeitsverteilungen zu erhalten.

e-bol.net

Darstellen der Wahrscheinlichkeiten

Um nicht raten zu müssen, welche Bereiche man ausprobieren sollte, können Sie eine grafische Darstellung der Wahrscheinlichkeitsdichte erstellen. Eine ausgezeichnete freie Bibliothek für das Erstellen mathematischer Graphen in Python ist *matplotlib*, die Sie von *http://matplotlib.sourceforge.net* herunterladen können.

Die Installationsanweisungen finden sich auf der Website, zudem erhalten Sie in Anhang A weitere Informationen zur *matplotlib*. Diese Bibliothek ist sehr mächtig und bietet viele Features, von denen nur ein paar in diesem Kapitel verwendet werden. Nach der Installation können Sie sie austesten, indem Sie einen einfachen Graphen in Ihrer Python-Session anlegen:

```
>>> from pylab import *
>>> a=array([1,2,3,4])
>>> b=array([4,2,3,1])
>>> plot(a,b)
>>> show()
>>> t1=arange(0.0,10.0,0.1)
>>> plot(t1,sin(t1))
[<matplotlib.lines.Line2D instance at 0x00ED9300>]
>>> show()
```

Damit sollten Sie einen Graphen erhalten, der so aussieht wie der in Abbildung 8-9. Die Funktion `arange` erstellt eine Liste mit Zahlen als Array – ähnlich wie die Funktion `range`. In diesem Fall zeichnen Sie eine Sinuskurve von 0 bis 10.

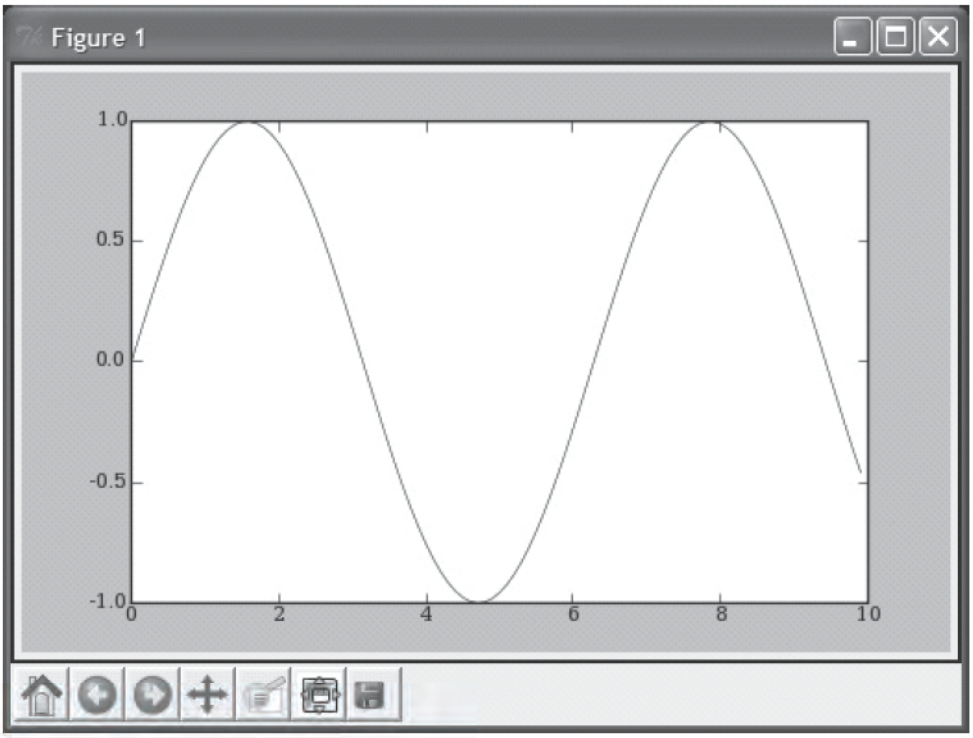

Abbildung 8-9: Beispiel für die Verwendung der Bibliothek matplotlib

e-bol.net

Dieser Abschnitt wird zwei verschiedene Varianten dazu vorstellen, wie man die Wahrscheinlichkeitsverteilung anzeigen kann. Die erste wird als *kumulative Wahrscheinlichkeit* bezeichnet. Ein Graph mit der kumulativen Wahrscheinlichkeit zeigt die Wahrscheinlichkeit, dass das Ergebnis kleiner als ein bestimmter Wert ist. Bei den Preisen beginnt der Graph mit Wahrscheinlichkeit 0, dass der Preis kleiner als 0 ist, und wächst an, wenn er bei bestimmten Preisen auf Elementgruppen trifft. Beim maximalen Preis erreicht der Graph die 1, da es eine 100%-Wahrscheinlichkeit gibt, dass der tatsächliche Preis kleiner oder gleich dem maximal möglichen Preis ist.

Um die Daten für die kumulative Wahrscheinlichkeit zu erstellen, muss man nur einen Preisbereich durchlaufen, der die Funktion `probabilityguess` mit 0 als untere Grenze und einem bestimmten Preis als obere Grenze aufruft. Das Ergebnis dieser Aufrufe kann an die Funktion `plot` übergeben werden, um einen Graphen zu zeichnen. Fügen Sie `cumulativegraph` der Datei *numpredict.py* hinzu:

```
def cumulativegraph(data,vec1,high,k=5,weightf=gaussian):
  t1=arange(0.0,high,0.1)
  cprob=array([probguess(data,vec1,0,v,k,weightf) for v in t1])
  plot(t1,cprob)
  show( )
```

Sie können diese Funktion nun in Ihrer Python-Session aufrufen, um den Graphen zu erstellen:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.cumulativegraph(data,(1,1),6)
```

Der Graph wird so aussehen wie der in Abbildung 8-10. Wie erwartet, beginnt die kumulative Wahrscheinlichkeit bei 0 und steigt zunehmend bis 1 an. Interessant am Graphen ist die Art, wie er steigt. Die Wahrscheinlichkeit bleibt bis etwa 50 Dollar bei 0 und steigt dann recht schnell an, verharrt erneut eine Weile bei 0,6 bis 110 Dollar und macht dort wieder einen Sprung.

Aus dem Graphen wird klar, dass die Wahrscheinlichkeiten um 60 Dollar und 110 Dollar herum gruppiert sind, da dort die kumulative Wahrscheinlichkeit einen Sprung macht. Weiß man das im Voraus, kann man die Wahrscheinlichkeitsberechnungen durchführen, ohne raten zu müssen.

Die andere Option ist, die tatsächlichen Wahrscheinlichkeiten für verschiedene Preise aufzutragen. Das ist schwieriger, da die Wahrscheinlichkeit eines ganz bestimmten Preises sehr gering ist. Würde man dieses aufzeichnen, hätte man fast überall 0 und nur bei den vorhergesagten Preisen eine Spitze. Stattdessen brauchen Sie eine Möglichkeit, Wahrscheinlichkeiten in bestimmten Bereichen zu kombinieren. So kann man zum Beispiel annehmen, dass die Wahrscheinlichkeit an jedem Punkt ein gewichteter Durchschnitt der umgebenden Wahrscheinlichkeiten ist – ähnlich wie der gewichtete kNN-Algorithmus.

e-bol.net

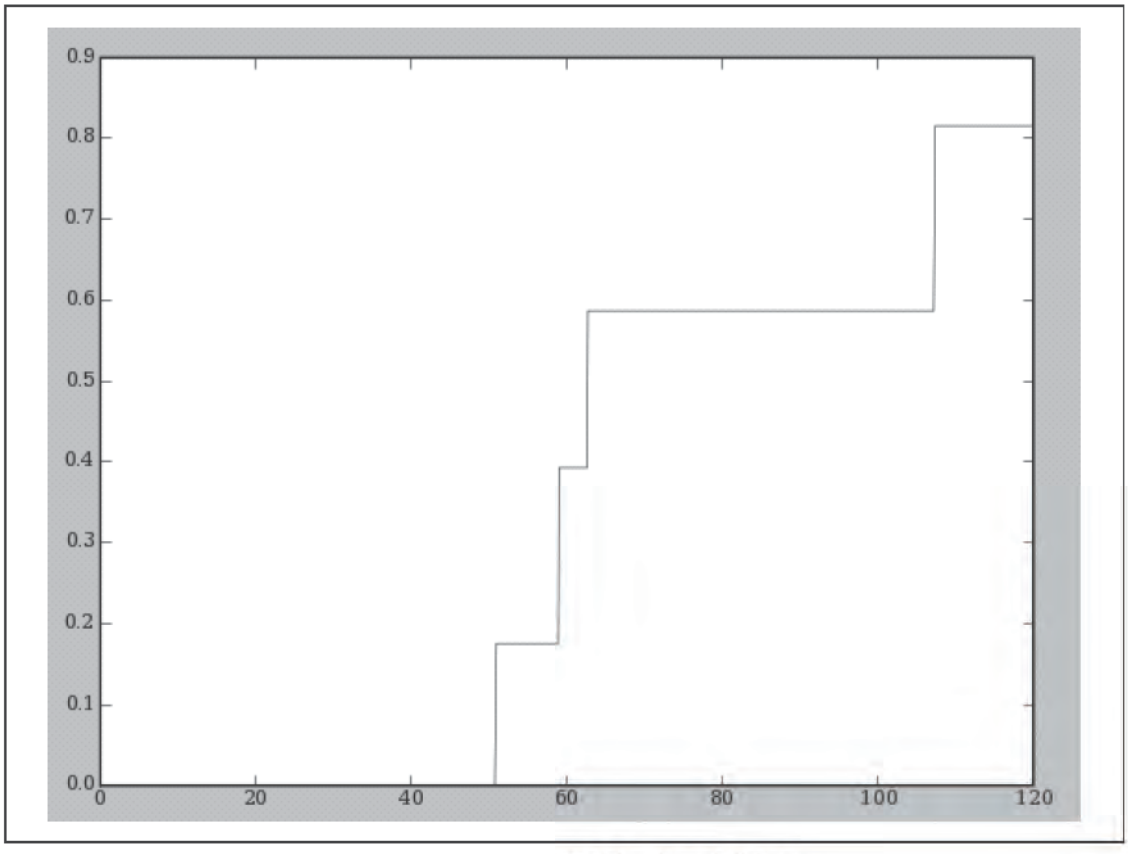

Abbildung 8-10: Ein Graph mit der kumulativen Wahrscheinlichkeit

Um das einmal selbst auszuprobieren, fügen Sie `probabilitygraph` der Datei *numpredict.py* hinzu:

```
def probabilitygraph(data,vec1,high,k=5,weightf=gaussian,ss=5.0):
  # Einen Bereich für die Preise erstellen.
  t1=arange(0.0,high,0.1)

  # Die Wahrscheinlichkeiten für den gesamten Bereich ermitteln.
  probs=[probguess(data,vec1,v,v+0.1,k,weightf) for v in t1]

  # Per Gauß mit den Nachbar-Wahrscheinlichkeiten glätten.
  smoothed=[]
  for i in range(len(probs)):
    sv=0.0
    for j in range(0,len(probs)):
      dist=abs(i-j)*0.1
      weight=gaussian(dist,sigma=ss)
      sv+=weight*probs[j]
    smoothed.append(sv)
  smoothed=array(smoothed)

  plot(t1,smoothed)
  show( )
```

e-bol.net

Diese Funktion erstellt einen Bereich von 0 bis high und berechnet dann die Wahrscheinlichkeiten für jeden Punkt. Da dies normalerweise sehr zerklüftet wäre, läuft die Funktion über das Array und erzeugt ein geglättetes Array, indem sie nahe Wahrscheinlichkeiten aufsummiert. Jeder Punkt der geglätteten Wahrscheinlichkeit ist die per Gauß gewichtete Summe seiner Nachbarn. Der Parameter ss legt fest, wie sehr die Wahrscheinlichkeiten geglättet werden sollen.

Testen Sie es nun in Ihrer Python-Session:

```
>>> reload(numpredict)
<module 'numpredict' from 'numpredict.py'>
>>> numpredict.probabilitygraph(data,(1,1),6)
```

Sie sollten einen Graph erhalten, der so ähnlich aussieht wie der in Abbildung 8-11.

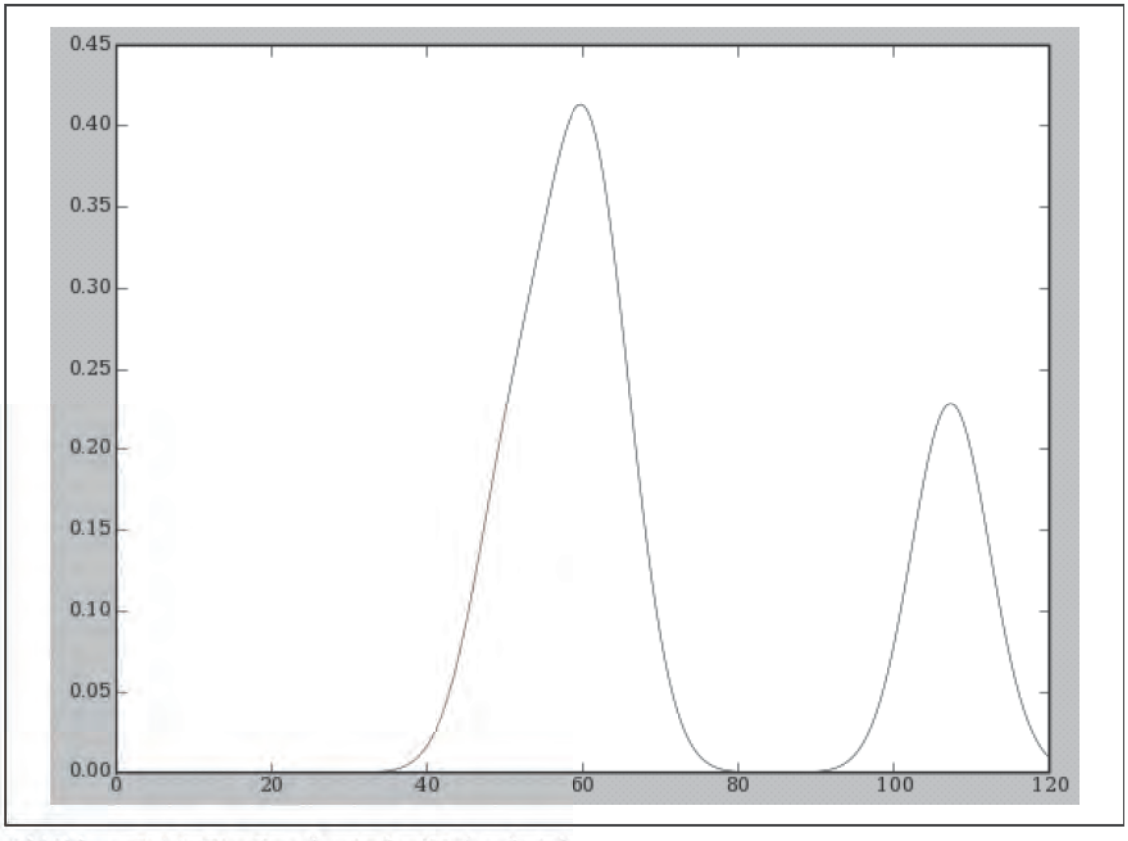

Abbildung 8-11: Ein Graph mit der Wahrscheinlichkeitsdichte

Dieser Graph macht es Ihnen noch leichter zu sehen, wo die Ergebnisse zu finden sind. Versuchen Sie einmal, das Fenster ss zu variieren, um herauszufinden, wie sich die Ausgabe ändert. Diese Wahrscheinlichkeitsverteilung macht deutlich, dass beim Abschätzen von Preisen für Wein ein wichtiger Aspekt in den Daten fehlt – nämlich, warum manche Personen bessere Preise erhalten als andere. In einigen Fällen werden Sie in der Lage sein herauszufinden, was da fehlt, aber bei anderen wer-

e-bol.net

den Sie feststellen, dass Sie sich einfach nach Objekten im unteren Preisbereich umsehen müssen.

Nutzung echter Daten – die eBay-API

eBay ist eine Online-Auktionssite und eine der beliebtesten Sites im Internet. Es gibt Millionen Produkte und Millionen Benutzer, die mitbieten und gemeinsam Preise festlegen, was ein tolles Beispiel für kollektive Intelligenz ist. Wie es der Zufall will, hat eBay auch eine freie XML-basierte API, mit der Sie Suchen durchführen, Detailinformationen zu Gegenständen erhalten und sogar selbst Angebote einstellen können. In diesem Abschnitt werden Sie lernen, wie Sie die eBay-API nutzen können, um Preisdaten zu erhalten, und diese dann so konvertieren, dass die Algorithmen in diesem Kapitel dazu genutzt werden können, Preise vorherzusagen.

Einen Entwicklerschlüssel bekommen

Um auf die eBay-API zugreifen zu können, muss man mehrere Schritte durchlaufen, aber der Vorgang ist recht einfach und automatisiert. Einen guten Überblick über den Prozess bietet der Quick Start Guide, den Sie unter *http://developer.ebay.com/quickstartguide* finden (auf Deutsch sind dies die ersten Schritte, die Sie unter *http://pages.ebay.de/entwickler/ersteSchritte.html* finden).

Dieser Guide wird Ihnen dabei helfen, einen Entwickler-Account zu erstellen, Ihre Schlüssel zu bekommen und ein Token anzulegen. Wenn Sie fertig sind, sollten Sie vier Strings haben, die für die Beispiele in diesem Kapitel benötigt werden:

- einen Entwickler-Key,
- einen Application-Key,
- einen Zertifikats-Key sowie
- ein Authentifizierungstoken, das sehr lang ist.

Erstellen Sie eine neue Datei namens *ebaypredict.py* und fügen Sie den folgenden Code ein, der ein paar Module importiert und die oben erwähnten Strings einbindet:

```
import httplib
from xml.dom.minidom import parse, parseString, Node

devKey = 'developerkey'
appKey = 'applicationkey'
certKey = 'certificatekey'
userToken = 'token'
serverUrl = 'api.ebay.com'
```

Es gibt keine offizielle Python-API für eBay, aber es gibt eine XML-API, auf die Sie per *httplib* und *minidom* zugreifen können. Dieser Abschnitt wird nur zwei Funktio-

e-bol.net

nen dieser API nutzen, `GetSearchResults` und `GetItem`, aber ein Großteil des hier vorgestellten Codes kann auch für andere Aufrufe verwendet werden. Mehr Informationen über alle von der API unterstützten Funktionen finden Sie in der vollständigen (englischen) Dokumentation unter *http://developer.ebay.com/DevZone/XML/docs/WebHelp/index.htm*.

Aufbauen einer Verbindung

Nachdem Sie Ihre Schlüssel haben, können Sie nun eine Verbindung zur eBay-API aufbauen. Die API erwartet von Ihnen die Übermittlung einer Reihe von Headern, die die Schlüssel und die Funktion beinhalten, die Sie nutzen wollen. Dazu erstellen Sie eine Funktion namens `getHeaders`, die den Funktionsnamen erwartet und ein Dictionary mit Headern zurückgibt, die an *httplib* übergeben werden können. Fügen Sie diese Funktion *ebaypredict.py* hinzu:

```
def getHeaders(apicall,siteID="0",compatabilityLevel = "433"):
  headers = {"X-EBAY-API-COMPATIBILITY-LEVEL": compatabilityLevel,
             "X-EBAY-API-DEV-NAME": devKey,
             "X-EBAY-API-APP-NAME": appKey,
             "X-EBAY-API-CERT-NAME": certKey,
             "X-EBAY-API-CALL-NAME": apicall,
             "X-EBAY-API-SITEID": siteID,
             "Content-Type": "text/xml"}
  return headers
```

Neben den Headern verlangt die eBay-API von Ihnen, XML mit Parametern für Ihren Request zu schicken. Sie liefert ein XML-Dokument zurück, das mit `parseString` aus der *minidom*-Bibliothek geparst werden kann.

Die Funktion, die den Request abschickt, öffnet eine Verbindung zum Server, postet den XML-Code mit den Parametern und parst das Ergebnis. Fügen Sie `sendrequest` der Datei *ebaypredict.py* hinzu:

```
def sendRequest(apicall,xmlparameters):
  connection = httplib.HTTPSConnection(serverUrl)
  connection.request("POST", '/ws/api.dll', xmlparameters, getHeaders(apicall))
  response = connection.getresponse( )
  if response.status != 200:
    print "Fehler beim Schicken des Requests: " + response.reason
  else:
    data = response.read( )
    connection.close( )
  return data
```

Diese Funktionen können genutzt werden, um Funktionen der eBay-API aufzurufen. Für die verschiedenen API-Aufrufe müssen Sie den XML-Code erzeugen und die geparsten Ergebnisse interpretieren können.

Da das Parsen des DOM nervig sein kann, sollten Sie eine einfache Hilfsmethode `getSingleValue` erstellen, die einen Knoten findet und seinen Inhalt zurückgibt:

e-bol.net

```
def getSingleValue(node,tag):
  nl=node.getElementsByTagName(tag)
  if len(nl)>0:
    tagNode=nl[0]
    if tagNode.hasChildNodes( ):
      return tagNode.firstChild.nodeValue
  return '-1'
```

Durchführen einer Suche

Um eine Suche durchzuführen, muss man lediglich die XML-Parameter für die API-Funktion `GetSearchResults` bauen und sie an die Funktion `sendrequest` übergeben, die Sie weiter oben definiert haben. Die XML-Parameter sehen so aus:

```
<GetSearchResultsRequest xmlns="urn:ebay:apis:eBLBaseComponents">
<RequesterCredentials><eBayAuthToken>token</eBayAuthToken></RequesterCredentials>
<parameter1>value</parameter1>
<parameter2>value</parameter2>
</GetSearchResultsRequest>
```

An diese API-Funktion können Dutzende von Parametern übergeben werden, aber in diesem Beispiel werden wir nur zwei nutzen:

`Query`
: Ein String mit den Suchbegriffen. Dieser Parameter wird genau so verwendet wie das Suchfeld auf der Homepage von eBay.

`CategoryID`
: Ein numerischer Wert, der die Kategorie angibt, in der Sie suchen wollen. eBay bietet eine umfangreiche Hierarchie mit Kategorien, die Sie mit der API-Funktion `GetCategories` ermitteln können. Dieser Parameter kann zusammen mit `Query` verwendet werden.

Die Funktion `doSearch` nimmt diese zwei Parameter und führt eine Suche durch. Sie liefert dann eine Liste mit Item-IDs zurück (die Sie später beim Aufruf von `GetItem` nutzen werden) und dazu deren Beschreibungen sowie die aktuellen Preise. Fügen Sie `doSearch` *ebaypredict.py* hinzu:

```
def doSearch(query,categoryID=None,page=1):
  xml = "<?xml version='1.0' encoding='utf-8'?>"+\
        "<GetSearchResultsRequest xmlns=\"urn:ebay:apis:eBLBaseComponents\">"+\
        "<RequesterCredentials><eBayAuthToken>" +\
        userToken +\
        "</eBayAuthToken></RequesterCredentials>" + \
        "<Pagination>"+\
          "<EntriesPerPage>200</EntriesPerPage>"+\
          "<PageNumber>"+str(page)+"</PageNumber>"+\
        "</Pagination>"+\
        "<Query>" + query + "</Query>"
  if categoryID!=None:
    xml+="<CategoryID>"+str(categoryID)+"</CategoryID>"
  xml+="</GetSearchResultsRequest>"
```

e-bol.net

```
  data=sendRequest('GetSearchResults',xml)
  response = parseString(data)
  itemNodes = response.getElementsByTagName('Item');
  results = []
  for item in itemNodes:
    itemId=getSingleValue(item,'ItemID')
    itemTitle=getSingleValue(item,'Title')
    itemPrice=getSingleValue(item,'CurrentPrice')
    itemEnds=getSingleValue(item,'EndTime')
    results.append((itemId,itemTitle,itemPrice,itemEnds))
  return results
```

Um den Parameter category zu nutzen, brauchen Sie auch noch eine Funktion, die die Hierarchie mit den Kategorien erstellt. Dafür gibt es eine weitere einfache API-Funktion, aber die XML-Datei für alle Kategorien ist sehr groß, braucht lange zum Herunterladen und lässt sich schlecht parsen. Aus diesem Grund sollten Sie die Kategoriedaten auf den Bereich beschränken, der Sie wirklich interessiert.

Die Funktion getCategory erwartet einen String und eine Parent-ID und liefert alle Kategorien zurück, die innerhalb dieser übergeordneten Kategorie den String enthalten. Wenn die Parent-ID fehlt, zeigt die Funktion einfach eine Liste aller Kategorien auf oberster Ebene an. Fügen Sie diese Funktion der Datei *ebaypredict.py* hinzu:

```
def getCategory(query='',parentID=None,siteID='0'):
  lquery=query.lower( )
  xml = "<?xml version='1.0' encoding='utf-8'?>"+\
        "<GetCategoriesRequest xmlns=\"urn:ebay:apis:eBLBaseComponents\">"+\
        "<RequesterCredentials><eBayAuthToken>" +\
        userToken +\
        "</eBayAuthToken></RequesterCredentials>"+\
        "<DetailLevel>ReturnAll</DetailLevel>"+\
        "<ViewAllNodes>true</ViewAllNodes>"+\
        "<CategorySiteID>"+siteID+"</CategorySiteID>"
  if parentID==None:
    xml+="<LevelLimit>1</LevelLimit>"
  else:
    xml+="<CategoryParent>"+str(parentID)+"</CategoryParent>"
  xml += "</GetCategoriesRequest>"
  data=sendRequest('GetCategories',xml)
  categoryList=parseString(data)
  catNodes=categoryList.getElementsByTagName('Category')
  for node in catNodes:
    catid=getSingleValue(node,'CategoryID')
    name=getSingleValue(node,'CategoryName')
    if name.lower( ).find(lquery)!=-1:
      print catid,name
```

Sie können nun diese Funktionen in Ihrer Python-Session austesten:

```
>>> import ebaypredict
>>> laptops=ebaypredict.doSearch('laptop')
>>> laptops[0:10]
```

e-bol.net

```
[(u'110075464522', u'Apple iBook G3 12" 500MHZ Laptop , 30 GB HD ', u'299.99',
u'2007-01-11T03:16:14.000Z'),
 (u'150078866214', u'512MB PC2700 DDR Memory 333MHz 200-Pin Laptop SODIMM', u'49.
99', u'2007-01-11T03:16:27.000Z'),
 (u'120067807006', u'LAPTOP USB / PS2 OPTICAL MOUSE 800 DPI SHIP FROM USA', u
'4.99', u'2007-01-11T03:17:00.000Z'),
...
```

Hoppla, das sieht ja so aus, als würde die Suche nach »laptop« alle möglichen Zubehörteile zurückliefern, die auch nur entfernt etwas mit Laptops zu tun haben. Glücklicherweise können Sie in der Kategorie »Laptops, Notebooks« suchen und Ihre Suche auf Ergebnisse beschränken, die tatsächlich Laptops sind. Sie brauchen dafür zunächst die Liste der obersten Kategorien und müssen dann innerhalb von »Computers and Networking« die Kategorie-ID für Laptops finden, um dann innerhalb dieser Kategorie nach »laptop« zu suchen:

```
>>> ebaypredict.getCategory('computers')
58058 Computers & Networking
>>> ebaypredict.getCategory('laptops',parentID=58058)
25447 Apple Laptops, Notebooks
...
31533 Drives for Laptops
51148 Laptops, Notebooks...
>>> laptops=ebaypredict.doSearch('laptop',categoryID=51148)
>>> laptops[0:10]
[(u'150078867562', u'PANASONIC TOUGHBOOK Back-Lit KeyBoard 4 CF-27 CF-28',
  u'49.95', u'2007-01-11T03:19:49.000Z'),
 (u'270075898309', u'mini small PANASONIC CFM33 CF M33 THOUGHBOOK ! libretto',
  u'171.0', u'2007-01-11T03:19:59.000Z'),
 (u'170067141814', u'Sony VAIO "PCG-GT1" Picturebook Tablet Laptop MINT  ',
  u'760.0', u'2007-01-11T03:20:06.000Z'),...
```

Zum Zeitpunkt der Entstehung dieses Buchs war die ID für die Kategorie »Laptops, Notebooks« die 51148. Sie werden feststellen, dass eine Beschränkung der Suche auf die Kategorie »iPod« viele der Ergebnisse unterdrückt, die nichts mit dem Thema zu tun haben, aber zu sehen wären, wenn Sie nur nach dem Begriff »iPod« suchen würden. Durch die höhere Konsistenz erhalten Sie eine bessere Datenmenge für ein Preismodell.

Details für ein Angebot erhalten

Die Ergebnisse der Suche liefern Titel und Preise zurück. Man kann vielleicht zusätzliche Details aus dem Text oder dem Titel extrahieren, wie zum Beispiel die Kapazität oder die Farbe. eBay stellt auch Attribute bereit, die für verschiedene Objekttypen spezifisch sind. Ein Laptop wird mit Attributen wie dem Prozessortyp und dem installierten RAM ausgegeben, während ein iPod Attribute wie zum Beispiel die Kapazität besitzt. Neben diesen Details ist es ebenfalls möglich, weitere Details zu bekommen, zum Beispiel die Bewertung des Verkäufers, die Anzahl der Gebote und den Startpreis.

e-bol.net

Um diese Details zu erhalten, müssen Sie die eBay-API-Funktion `GetItem` aufrufen und die ID des Objekts übergeben, die die Suchfunktion geliefert hat. Dazu erstellen Sie eine Funktion namens `getItem` in *ebaypredict.py*:

```
def getItem(itemID):
  xml = "<?xml version='1.0' encoding='utf-8'?>"+\
        "<GetItemRequest xmlns=\"urn:ebay:apis:eBLBaseComponents\">"+\
        "<RequesterCredentials><eBayAuthToken>" +\
        userToken +\
        "</eBayAuthToken></RequesterCredentials>" + \
        "<ItemID>" + str(itemID) + "</ItemID>"+\
        "<DetailLevel>ItemReturnAttributes</DetailLevel>"+\
        "</GetItemRequest>"
  data=sendRequest('GetItem',xml)
  result={}
  response=parseString(data)
  result['title']=getSingleValue(response,'Title')
  sellingStatusNode = response.getElementsByTagName('SellingStatus')[0];
  result['price']=getSingleValue(sellingStatusNode,'CurrentPrice')
  result['bids']=getSingleValue(sellingStatusNode,'BidCount')
  seller = response.getElementsByTagName('Seller')
  result['feedback'] = getSingleValue(seller[0],'FeedbackScore')

  attributeSet=response.getElementsByTagName('Attribute');
  attributes={}
  for att in attributeSet:
    attID=att.attributes.getNamedItem('attributeID').nodeValue
    attValue=getSingleValue(att,'ValueLiteral')
    attributes[attID]=attValue
  result['attributes']=attributes
  return result
```

Diese Funktion holt den XML-Code des Objekts mit `sendrequest` und extrahiert die interessanten Daten. Da die Attribute für jedes Objekt unterschiedlich sind, werden alle in einem Dictionary zurückgegeben. Probieren Sie diese Funktion mit einem der Ergebnisse aus, die Sie durch Ihre Suche erhalten haben:

```
>>> reload(ebaypredict)
>>> ebaypredict.getItem(laptops[7][0])
{'attributes': {u'13': u'Windows XP', u'12': u'512', u'14': u'Compaq',
                u'3805': u'Exchange', u'3804': u'14 Days',
                u'41': u'-', u'26445': u'DVD+/-RW', u'25710': u'80.0',
                u'26443': u'AMD Turion 64', u'26444': u'1800', u'26446': u'15',
                u'10244': u'-'},
'price': u'515.0', 'bids': u'28', 'feedback': u'2797',
'title': u'COMPAQ V5210US 15.4" AMD Turion 64 80GB Laptop Notebook'}
```

Hier kann man erkennen, dass das Attribut 26444 anscheinend die Prozessorgeschwindigkeit widerspiegelt, 26446 für die Bildschirmgröße steht, 12 für den installierten Speicher und 25710 für die Größe der Festplatte. Zusammen mit der Verkäuferbewertung, der Anzahl der Gebote und dem Startpreis ergibt sich so eine potenziell interessante Datenmenge für Preisvorhersagen.

e-bol.net

Aufbau einer Preisprognose

Um die Prognosealgorithmen aus diesem Kapitel nutzen zu können, brauchen Sie diverse Angebote von eBay und müssen diese in Listen mit Zahlen umwandeln, die als Datenmenge für die Kreuzvalidierungsfunktion genutzt werden können. Dazu ruft die Funktion `makeLaptopDataset` zunächst `doSearch` auf, um eine Liste mit Laptops zu erhalten, und fragt dann jeden einzelnen ab. Mit den im vorigen Abschnitt gefundenen Attributen erstellt die Funktion eine Liste mit Zahlen, die für die Vorhersage genutzt werden können, und füllt die Strukturen für die kNN-Funktionen.

Fügen Sie `makeLaptopDataset` der Datei *ebaypredict.py* hinzu:

```
def makeLaptopDataset( ):
  searchResults=doSearch('laptop',categoryID=51148)
  result=[]
  for r in searchResults:
    item=getItem(r[0])
    att=item['attributes']
    try:
      data=(float(att['12']),float(att['26444']),
            float(att['26446']),float(att['25710']),
            float(item['feedback'])
           )
      entry={'input':data,'result':float(item['price'])}
      result.append(entry)
    except:
      print item['title']+' failed'
  return result
```

Die Funktion ignoriert alle Objekte, die nicht die notwendigen Attribute haben. Das Herunterladen und Verarbeiten aller Ergebnisse dauert einige Zeit, aber dann haben Sie eine interessantes Datenmenge mit echten Preisen und Attributen, mit denen Sie spielen können. Um an die Daten heranzukommen, rufen Sie die Funktion in Ihrer Python-Session auf:

```
>>> reload(ebaypredict)
<module 'ebaypredict' from 'ebaypredict.py'>
>>> set1=ebaypredict.makeLaptopDataset( )
...
```

Sie können nun ein paar kNN-Abschätzungen mit verschiedenen Konfigurationen ausprobieren:

```
>>> numpredict.knnestimate(set1,(512,1000,14,40,1000))
667.89999999999998
>>> numpredict.knnestimate(set1,(1024,1000,14,40,1000))
858.42599999999982
>>> numpredict.knnestimate(set1,(1024,1000,14,60,0))
482.02600000000001
>>> numpredict.knnestimate(set1,(1024,2000,14,60,1000))
1066.8
```

e-bol.net

Dies zeigt ein wenig den Einfluss verschiedener Speichergrößen, Prozessorgeschwindigkeiten und Verkäuferbewertungen. Jetzt können Sie versuchen, mit verschiedenen Variablen zu experimentieren, die Daten zu skalieren und die Wahrscheinlichkeitsverteilungen auszugeben.

Wann man k-nächste Nachbarn nutzt

Die Methode der k-nächsten Nachbarn hat ein paar Nachteile. Das Erstellen von Vorhersagen ist sehr rechenintensiv, da die Abstände zu jedem Punkt berechnet werden müssen. Zudem kann es in einer Datenmenge mit vielen Variablen schwierig sein, die passenden Gewichtungen zu bestimmen oder herauszufinden, ob bestimmte Variablen ausgemerzt werden sollten. Dabei kann eine Optimierung helfen, aber es ist durchaus möglich, dass es eine sehr lange Zeit dauert, bis bei großen Datenmengen eine gute Lösung gefunden wurde.

Aber wie Sie in diesem Kapitel gesehen haben, bietet kNN trotzdem eine Reihe von Vorteilen gegenüber anderen Methoden. Die Kehrseite des Rechenaufwands beim Erstellen einer Vorhersage ist, dass den Daten neue Beobachtungen hinzugefügt werden können, ohne neue Berechnungen anstellen zu müssen. Es ist zudem einfach, genau zu interpretieren, was geschieht, da Sie wissen, dass die gewichteten Werte anderer Beobachtungen genutzt werden.

Auch wenn das Bestimmen der Gewichtungen schwierig sein kann, können Sie sie nutzen, um die Charakteristiken der Datenmenge besser zu verstehen, sobald einmal die besten Gewichtungen gefunden wurden. Schließlich können Sie Wahrscheinlichkeitsfunktionen erstellen für den Fall, dass Sie den Verdacht haben, es gäbe andere, bisher nicht erfasste Variablen in der Datenmenge.

Übungen

1. *Optimieren der Anzahl der Nachbarn*. Erstellen Sie eine Zielfunktion für die Optimierung, die die ideale Anzahl an Nachbarn für einr einfache Datenmenge bestimmt.
2. *Leave-One-Out-Kreuzvalidierung*. Die Leave-One-Out-Kreuzvalidierung ist eine alternative Methode zum Berechnen des Vorhersagefehlers, die jede Zeile in der Datenmenge einzeln als Testset nutzt und den Rest der Daten als Trainingsset verwendet. Implementieren Sie dafür eine Funktion. Wie verhält sie sich zur weiter oben im Kapitel beschriebenen Kreuzvalidierungsfunktion?
3. *Eliminieren von Variablen*. Anstatt zu versuchen, die Skalierungen der Variablen per Optimierung zu bestimmen, die möglicherweise nutzlos sind, könnten Sie versuchen, Variablen gleich zu Beginn zu eliminieren, die die Vorhersage viel schlechter machen. Können Sie sich einen Weg dafür vorstellen?

e-bol.net

4. *Variieren von ss für die Darstellung der Wahrscheinlichkeit*. Der Parameter `ss` in `probabilityguess` gibt vor, wie glatt die Wahrscheinlichkeit gezeichnet wird. Was passiert, wenn der Wert zu groß ist? Oder zu niedrig? Können Sie sich eine Möglichkeit vorstellen, einen guten Wert herauszufinden, ohne den Graphen zu betrachten?
5. *Das Laptop-Datenmenge*. Versuchen Sie, die Optimierung für die Laptop-Datenmenge von eBay durchzuführen. Welche Variablen sind wichtig? Anschließend lassen Sie die Wahrscheinlichkeitsdichte zeichnen. Gibt es beachtenswerte Spitzen?
6. *Andere Element-Typen*. Welche anderen Objekte bei eBay haben nützliche numerische Attribute? iPods, Handys und Autos bieten allesamt eine Reihe nützlicher Informationen an. Versuchen Sie, eine weitere Datenmenge zur numerischen Vorhersage zu erstellen.
7. *Suchen nach Attributen*. Die eBay-API stellt eine ganze Reihe von Funktionen bereit, die nicht in diesem Kapitel genutzt wurden. Die Funktion `GetSearchResults` hat viele Optionen, einschließlich der Fähigkeit, Suchen auf bestimmte Attribute einzuschränken. Passen Sie Ihre Funktion so an, dass sie dies unterstützt, und versuchen Sie, nur Core-Duo-Laptops zu finden.

e-bol.net

KAPITEL 9

Komplexe Klassifikation: Kernel-Methoden und SVMs

In den vorigen Kapiteln wurden verschiedene Klassifizierer vorgestellt: Entscheidungsbäume, Bayes-Klassifizierer und neuronale Netze. Dieses Kapitel wird das Konzept der linearen Klassifizierer und Kernel-Methoden behandeln – als Vorspiel zu einem der komplexesten Klassifizierern, einem Bereich, in dem immer noch aktiv geforscht wird: den *Support-Vektor-Maschinen* (SVMs).

Die Datenmenge, die in einem Großteil dieses Kapitels verwendet wird, dreht sich darum, Personen auf einer Dating-Site zusammenzubringen. Können wir anhand der Informationen über zwei Personen voraussagen, ob sie gut zusammenpassen? Das ist ein interessantes Problem, da es viele Variablen gibt – sowohl numerische wie auch nominelle – und viele nicht lineare Beziehungen. Diese Datenmenge wird im Folgenden genutzt, um ein paar der Schwächen der bisher beschriebenen Klassifizierer aufzuzeigen, und um zeigen, wie die Datenmenge angepasst werden kann, damit sie besser für diese Algorithmen geeignet ist. Was man aus diesem Kapitel mitnehmen sollte, ist, dass es sehr unwahrscheinlich ist, eine komplexe Datenmenge auf solch einen Algorithmus zu werfen und zu erwarten, dass dieser lernt, die Dinge genau klassifizieren zu können. Um gute Ergebnisse zu erhalten, muss man häufig den richtigen Algorithmus wählen und die Daten passend vorbereiten. Ich hoffe, dass Sie aus dem Anpassen dieser Datenmenge lernen, wie Sie selbst andere Daten in Zukunft präparieren können.

Am Ende dieses Kapitels werden Sie erfahren, wie Sie eine Datenmenge mit echten Personen aus *Facebook* aufbauen (einer beliebten Social Networking Site) und die Algorithmen nutzen, um vorauszusagen, ob Personen mit bestimmten Charakteristiken Freunde werden können.

Datenmenge für Vermittler

Die Datenmenge, die Sie in diesem Kapitel verwenden werden, basiert auf einer imaginären Online-Dating-Site. Die meisten Dating-Sites sammeln eine Reihe inter-

e-bol.net

essanter Informationen über ihre Mitglieder, so zum Beispiel demografische Daten, Interessen und Verhaltensweisen. Stellen Sie sich vor, dass diese Site die folgenden Informationen speichert:

- Alter
- Raucher?
- Möchte Kinder haben?
- Interessenliste
- Standort

Zudem merkt sich die Site Informationen dazu, ob zwei Personen gut zusammenpassten, ob sie einen ersten Kontakt gewagt haben und ob sie sich dazu entschlossen haben, sich persönlich zu treffen. Diese Daten werden genutzt, um die Vermittler-Datenmenge zu erstellen. Es gibt zwei Dateien, die heruntergeladen werden können:

http://kiwitobes.com/matchmaker/agesonly.csv
http://kiwitobes.com/matchmaker/matchmaker.csv

Die Datei *matchmaker.csv* sieht so aus:

```
39,yes,no,skiing:knitting:dancing,220 W 42nd St New York
 NY,43,no,yes,soccer:reading:scrabble,824 3rd Ave New York NY,0
23,no,no,football:fashion,102 1st Ave New York
 NY,30,no,no,snowboarding:knitting:computers:shopping:tv:travel,
 151 W 34th St New York NY,1
50,no,no,fashion:opera:tv:travel,686 Avenue of the Americas
 New York NY,49,yes,yes,soccer:fashion:photography:computers:
 camping:movies:tv,824 3rd Ave New York NY,0
```

Jede Zeile enthält Informationen über einen Mann und eine Frau und in der letzten Spalte eine 1 oder 0, um anzugeben, ob davon ausgegangen wird, dass sie zusammenpassen. (Der Autor ist sich durchaus bewusst, dass hier vereinfachende Annahmen getroffen wurden – Computermodelle sind niemals so kompliziert wie das wahre Leben.) Bei einer Site mit einer großen Anzahl an Profilen könnten diese Informationen genutzt werden, um einen Vorhersagealgorithmus aufzubauen, der Benutzer dabei unterstützt, andere Personen zu finden, die vermutlich zu ihm passen. Zudem können damit bestimmte Personentypen ermittelt werden, die auf der Site nur wenig vorhanden sind, wodurch man Strategien entwickeln kann, um neue Mitglieder zu gewinnen. Die Datei *agesonly.csv* bietet Kuppelinformationen nur auf Basis des Alters. Wir werden sie nutzen, um zu zeigen, wie der Klassifizierer funktioniert, da sich zwei Variablen viel einfacher darstellen lassen.

Der erste Schritt besteht darin, eine Funktion zum Laden dieser Datenmenge zu erstellen. Dazu müssen nur alle Felder in eine Liste geladen werden, aber zum Ausprobieren wird die Funktion einen optionalen Parameter haben, um lediglich bestimmte Felder zu laden. Erstellen Sie eine neue Datei namens *advancedclassify.py* und fügen Sie die Klasse `matchrow` sowie die Funktion `loadmatch` hinzu:

e-bol.net

```
class matchrow:
  def __init__(self,row,allnum=False):
    if allnum:
      self.data=[float(row[i]) for i in range(len(row)-1)]
    else:
      self.data=row[0:len(row)-1]
    self.match=int(row[len(row)-1])

def loadmatch(f,allnum=False):
  rows=[]
  for line in file(f):
    rows.append(matchrow(line.split(','),allnum))
  return rows
```

`loadmatch` erzeugt eine Liste mit Klassen des Typs `matchrow`, wobei jede die eigentlichen Daten sowie die Information, ob dies ein Treffer war, enthält. Verwenden Sie diese Funktion, um sowohl die Datenmenge nur mit dem Alter als auch die vollständige Kuppel-Datenmenge zu laden:

```
>>> import advancedclassify
>>> agesonly=advancedclassify.loadmatch('agesonly.csv',allnum=True)
>>> matchmaker=advancedclassify.loadmatch('matchmaker.csv')
```

Schwierigkeiten mit den Daten

Zwei interessante Aspekte dieser Datenmenge sind die Nichtlinearität und die Wechselwirkungen zwischen den Variablen. Wenn Sie in Kapitel 8 die *matplotlib* (*http://matplotlib.sourceforge.net*) installiert haben, können Sie ein paar der Variablen mit `advancedclassify` auftragen lassen und einige Listen daraus erzeugen. (Dieser Schritt ist nicht notwendig, um mit dem Rest des Kapitels fortfahren zu können.) Probieren Sie dies einmal in Ihrer Python-Session aus:

```
from pylab import *
def plotagematches(rows):
  xdm,ydm=[r.data[0] for r in rows if r.match==1],\
          [r.data[1] for r in rows if r.match==1]
  xdn,ydn=[r.data[0] for r in rows if r.match==0],\
          [r.data[1] for r in rows if r.match==0]

  plot(xdm,ydm,'go')
  plot(xdn,ydn,'ro')

  show()
```

Rufen Sie diese Methode in Ihrer Python-Session auf:

```
>>> reload(advancedclassify)
<module 'advancedclassify' from 'advancedclassify.py'>
>>> advancedclassify.plotagematches(agesonly)
```

Damit erhält man eine *Punktwolke* mit dem Alter der Männer im Vergleich zu dem der Frauen. Die Punkte sind ein O, wenn die Personen passen, und ein X, wenn sie

e-bol.net

nicht passen. Sie werden ein Fenster erhalten, das so aussieht wie das in Abbildung 9-1.

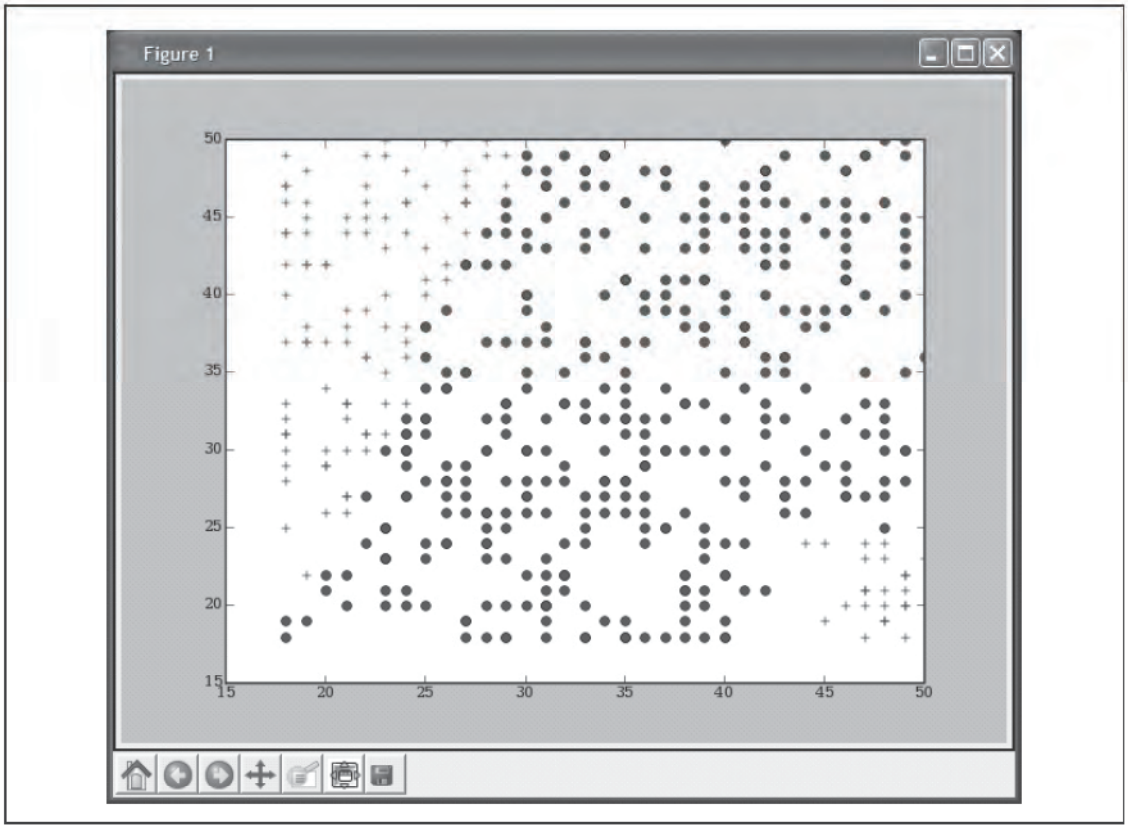

Abbildung 9-1: Generierte Alter-zu-Alter-Punktwolke

Obwohl es natürlich viele andere Faktoren gibt, die entscheiden, ob zwei Personen zusammenpassen, basiert diese Abbildung auf der vereinfachten Datenmenge nur mit den Altersangaben. Sie macht eine deutliche Grenze sichtbar, die zeigt, dass die Menschen bei der Partnerwahl nicht sehr weit von ihrem eigenen Altersbereich abweichen. Die Grenze verändert sich und wird weniger exakt, sobald die Menschen älter werden, was zeigt, dass man mit zunehmendem Alter größere Altersunterschiede toleriert.

Entscheidungsbaum-Klassifizierer

Kapitel 7 drehte sich um Entscheidungsbaum-Klassifizierer, die versuchen, Daten automatisch mithilfe eines Baums zu klassifizieren. Der dort beschriebene Entscheidungsbaum-Algorithmus teilt die Daten anhand numerischer Grenzen auf. Das führt zu einem Problem, wenn die Trennlinie besser als eine Funktion von zwei Variablen beschrieben werden kann. In diesem Fall ist die Differenz zwischen den beiden Altern eine deutlich bessere Variable für die Voraussage. Sie können sich sicher vorstellen, dass ein Training des Entscheidungsbaums direkt mit den Daten zu einem Ergebnis führen würde, das so aussieht wie das in Abbildung 9-2.

e-bol.net

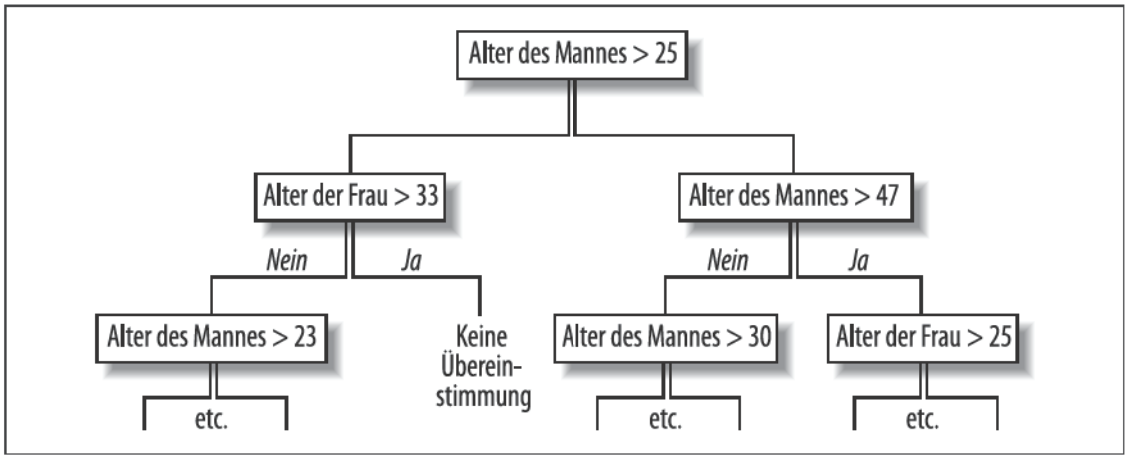

Abbildung 9-2: Entscheidungsbaum für eine nicht geradlinige Grenze

Das ist für eine Interpretation offensichtlich ziemlich nutzlos. Es mag für eine automatische Klassifikation ausreichen, aber es ist sehr unübersichtlich und zu streng. Würde man jetzt auch noch neben dem Alter die anderen Variablen berücksichtigen, würde der Baum noch verwirrender werden. Um zu verstehen, was der Entscheidungsbaum tut, schauen Sie sich die Punktwolke und die *Entscheidungsgrenze* (*Decision Boundary*) an, die vom Entscheidungsbaum erstellt wurde (siehe Abbildung 9-3).

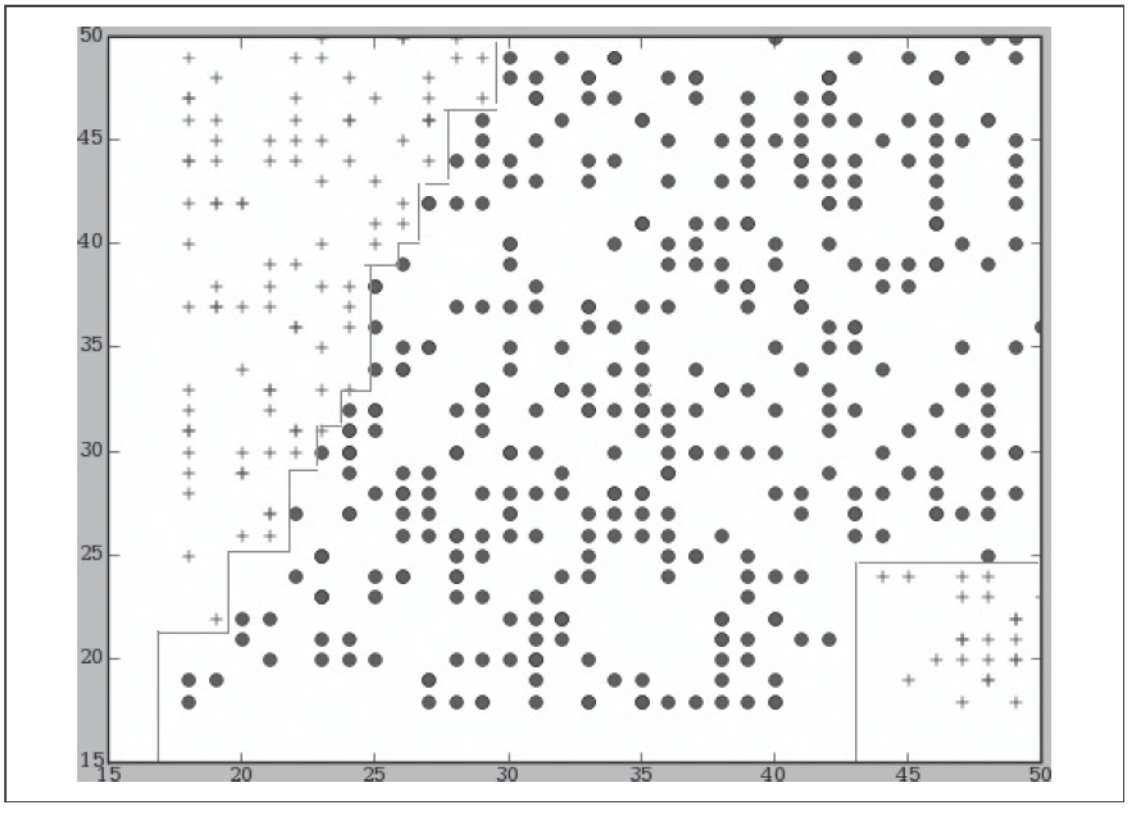

Abbildung 9-3: Durch einen Entscheidungsbaum erzeugte Grenze

e-bol.net

Die Entscheidungsgrenze ist die Linie, bei der jeder Punkt auf der einen Seite der einen Kategorie und jeder Punkt auf der anderen Seite der anderen Kategorie zugewiesen wird. Die Abbildung macht deutlich, dass die Einschränkungen des Entscheidungsbaums die Grenzen dazu zwingen, entweder horizontal oder vertikal zu verlaufen.

Es gibt dazu zwei wichtige Hinweise. Zum einen ist es keine gute Idee, die Daten einfach ganz naiv so zu verwenden, ohne sich Gedanken darüber zu machen, was sie bedeuten und wie sie so umgewandelt werden können, dass sie sich leichter interpretieren lassen. Wenn man einen Punktwolken-Graph erstellt, kann das dabei helfen herauszufinden, wie die Daten wirklich aufgeteilt sind. Zum anderen haben Entscheidungsbäume zwar ihre Stärken, die in Kapitel 7 beschriebenen Bäume sind aber meist nur eine sehr schlechte Möglichkeit, die Klassen bei Problemen zu bestimmen, die viele numerische Eingabewerte ohne einfache Beziehungen untereinander enthalten.

Einfache lineare Klassifikation

Dies ist einer der am einfachsten zu erstellenden Klassifizierer, aber eine gute Grundlage für die weitere Arbeit. Er ermittelt den Durchschnitt aller Daten in jeder Klasse und legt einen Punkt fest, der den Mittelpunkt dieser Klasse definiert. Dann kann er neue Punkte klassifizieren, indem er ermittelt, welchem Mittelpunkt sie am nächsten sind.

Dazu benötigen Sie zunächst eine Funktion, die den *Mittelpunkt* jeder Klasse errechnet. In diesem Fall sind die Klassen nur 0 und 1. Fügen Sie `lineartrain` *advancedclassify.py* hinzu:

```
def lineartrain(rows):
  averages={}
  counts={}

  for row in rows:
    # Klasse dieses Punkts ermitteln.
    cl=row.match

    averages.setdefault(cl,[0.0]*(len(row.data)))
    counts.setdefault(cl,0)

    # Diesen Punkt zum Durchschnitt hinzufügen.
    for i in range(len(row.data)):
      averages[cl][i]+=float(row.data[i])

    # Mitzählen, wie viele Punkte in jeder Klasse sind.
    counts[cl]+=1

  # Summen durch die Anzahl teilen, um die Durchschnitte zu erhalten.
  for cl,avg in averages.items( ):
```

e-bol.net

```
    for i in range(len(avg)):
      avg[i]/=counts[cl]

  return averages
```

Sie können diese Funktion in Ihrer Python-Session ausführen, um die Durchschnittswerte zu erhalten:

```
>>> reload(advancedclassify)
<module 'advancedclassify' from 'advancedclassify.pyc'>
>>> avgs=advancedclassify.lineartrain(agesonly)
```

Damit Sie verstehen, warum das nützlich ist, schauen Sie sich noch einmal den Graphen mit den Altersdaten an (Abbildung 9-4).

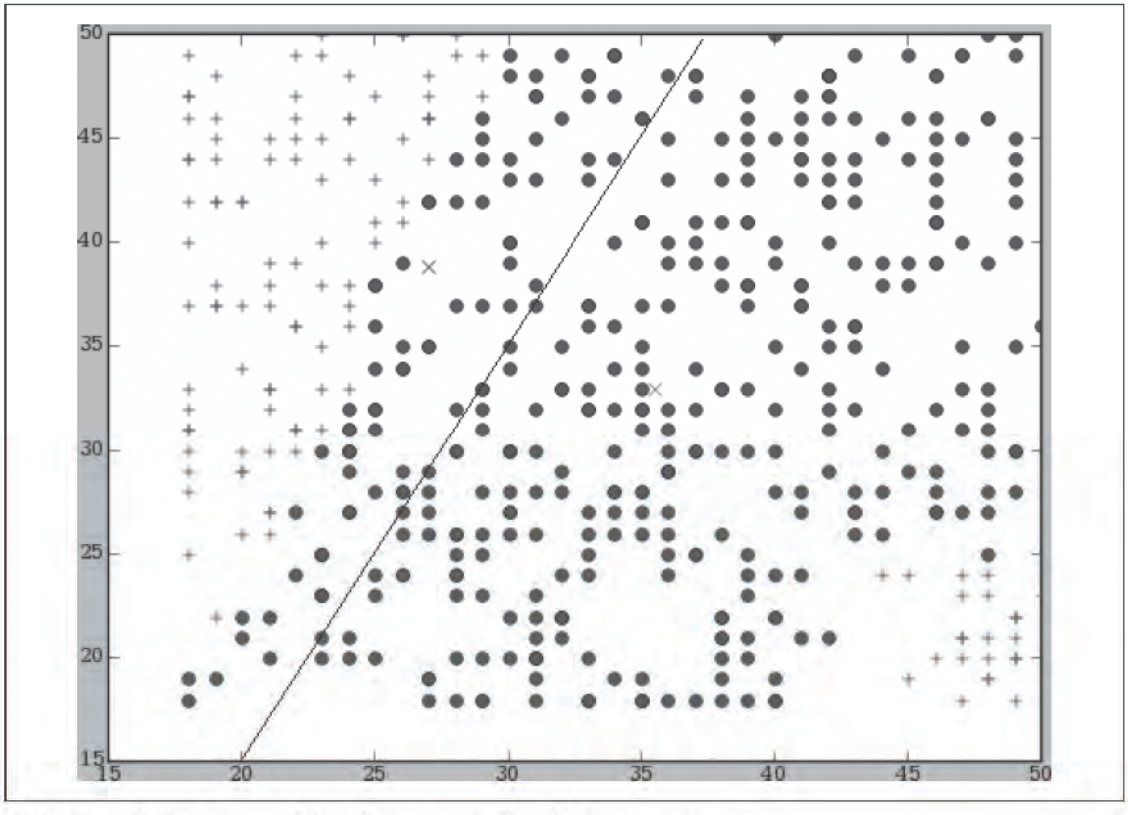

Abbildung 9-4: Linearer Klassifizierer mit Durchschnittswerten

Die X-Zeichen in dieser Abbildung repräsentieren die Durchschnittspunkte, die von `lineartrain` berechnet wurden. Die Linie, die die Daten aufteilt, liegt genau in der Mitte zwischen den beiden X. Das bedeutet, dass alle Punkte auf der linken Seite der Linie näher am Mittelpunkt von »passen nicht« und alle Punkte rechts von der Linie näher am Mittelpunkt von »passen zusammen« liegen. Immer wenn Sie ein neues Alterspaar haben und herausfinden wollen, ob zwei Personen zusammenpassen, können Sie den Punkt in diesen Graph eintragen und schauen, welcher Mittelpunkt näher liegt.

e-bol.net

Es gibt viele Wege herauszufinden, wie nah ein neuer Punkt ist. Sie haben in den bisherigen Kapiteln schon den euklidischen Abstand kennengelernt. Eine Möglichkeit wäre, den Abstand des Punkts zu jedem Mittelpunkt der Klassen zu berechnen und die Klasse zu wählen, bei der der Abstand am kleinsten ist. Das funktioniert zwar bei diesem einfachen Klassifizierer, aber um später noch Erweiterungen vornehmen zu können, müssen Sie anders vorgehen – mit *Vektoren* und *Skalarprodukten*.

Ein Vektor hat eine Größe und eine Richtung und wird häufig als Pfeil in einer Ebene gezeichnet oder als Reihe von Zahlen geschrieben. Abbildung 9-5 zeigt ein Beispiel eines Vektors. Die Abbildung macht auch deutlich, wie man einen Vektor zwischen zwei Punkten erhält, wenn man den einen vom anderen abzieht.

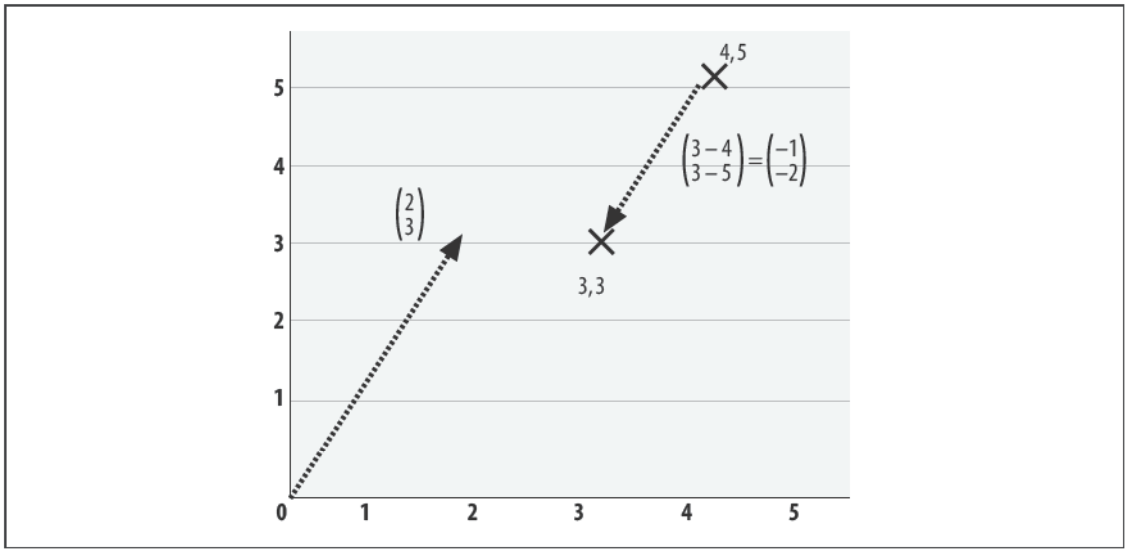

Abbildung 9-5: Vektorbeispiele

Ein Skalarprodukt (oder auch inneres Produkt oder Punktprodukt, im Englischen *Dot Product*) liefert eine einzelne Zahl (ein Skalar) als Ergebnis von zwei Vektoren, indem jeder Wert des ersten Vektors mit dem entsprechenden Wert des zweiten multipliziert und alle addiert werden. Erstellen Sie eine neue Funktion namens `dotproduct` in *advancedclassify.py*:

```
def dotproduct(v1,v2):
  return sum([v1[i]*v2[i] for i in range(len(v1))])
```

Das Skalarprodukt entspricht auch dem Produkt der Längen der beiden Vektoren, multipliziert mit dem Kosinus des Winkels zwischen ihnen. Wichtig ist, dass der Kosinus eines Winkels negativ wird, wenn der Winkel größer als 90° ist, was bedeutet, dass das Skalarprodukt in diesen Fällen ebenfalls negativ ist. Um zu sehen, was für einen Vorteil Sie daraus ziehen können, schauen Sie sich Abbildung 9-6 an.

In diesem Diagramm sehen Sie jeweils einen Mittelpunkt für »passen zusammen« (M_0) und »passen nicht zusammen« (M_1) sowie C, der genau in der Mitte zwischen

e-bol.net

beiden Punkten liegt. Es gibt noch zwei weitere Punkte X_0 und X_1, die zu klassifizierende Beispiele sind. Der Vektor, der von M_0 zu M_1 verläuft, ist eingezeichnet, ebenso die Vektoren, die von X_1 und X_2 zu C verlaufen.

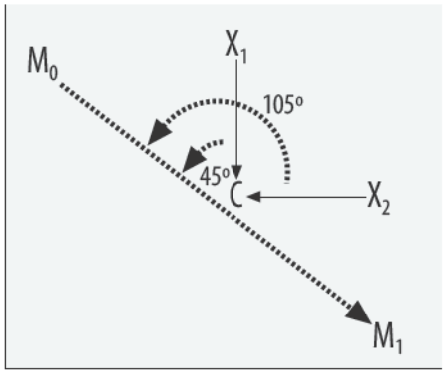

Abbildung 9-6: Verwendung des Skalarprodukts, um den Abstand zu bestimmen

In dieser Abbildung liegt X_1 näher an M_0, daher sollte es als »passen zusammen« klassifiziert werden. Sie sehen, dass der Winkel zwischen den Vektoren $X_1 \rightarrow C$ und $M_0 \rightarrow M_1$ 45° beträgt, was weniger als 90° sind, daher ist das Skalarprodukt von $X_1 \rightarrow C$ und $M_0 \rightarrow M_1$ positiv.

Andererseits ist der Winkel zwischen $X_2 \rightarrow C$ und $M_0 \rightarrow M_1$ größer als 90°, da die Vektoren in entgegengesetzte Richtungen zeigen. Das bedeutet, dass das Skalarprodukt von $X_2 \rightarrow C$ und $M_0 \rightarrow M_1$ negativ ist.

Das Skalarprodukt ist für große Winkel negativ und für kleine Winkel positiv, daher brauchen Sie nur auf das Vorzeichen des Skalarprodukts zu schauen, um herauszufinden, zu welcher Klasse der neue Punkt gehört.

Der Punkt C ist der Mittelwert von M_0 und M_1 oder auch $(M_0+M_1)/2$, daher ist die Formel zum Finden der Klasse:

$$class=sign((X - (M_0+M_1)/2) \cdot (M_0-M_1))$$

Multipliziert man das aus, erhält man:

$$class=sign(X.M_0 - X.M_1 + (M_0.M_0 - M_1.M_1)/2)$$

Dies ist die Formel, die wir nutzen werden, um die Klasse zu bestimmen. Fügen Sie eine neue Funktion namens `dpclassify` der Datei *advancedclassify.py* hinzu:

```
def dpclassify(point,avgs):
  b=(dotproduct(avgs[1],avgs[1])-dotproduct(avgs[0],avgs[0]))/2
  y=dotproduct(point,avgs[0])-dotproduct(point,avgs[1])+b
  if y>0: return 0
  else: return 1
```

Jetzt können Sie den Klassifizierer nutzen, um ein paar Ergebnisse für die Daten in Ihrer Python-Session zu erhalten:

e-bol.net

```
>>> reload(advancedclassify)
<module 'advancedclassify' from 'advancedclassify.py'>
>>> advancedclassify.dpclassify([30,30],avgs)
1
>>> advancedclassify.dpclassify([30,25],avgs)
1
>>> advancedclassify.dpclassify([25,40],avgs)
0
>>> advancedclassify.dpclassify([48,20],avgs)
1
```

Denken Sie daran, dass dies ein linearer Klassifizierer ist, daher findet er nur eine Trennlinie. Das bedeutet, dass der Klassifizierer nicht immer die richtigen Antworten liefern wird, wenn es keine gerade Linie zwischen den Datengruppen gibt oder mehrere Bereiche vorhanden sind, wie dies beim Alter-Alter-Vergleich der Fall ist. In diesem Beispiel sollte der Vergleich der Altersangaben von 48 und 20 wirklich zu einem »passen nicht zusammen« führen, aber da nur eine Linie gefunden wurde und der Punkt auf der rechten Seite davon liegt, hat sich die Funktion dazu entschieden, dass die beiden Personen passen. Im später folgenden Abschnitt »Kernel-Methoden verstehen« werden Sie sehen, wie Sie diesen Klassifizierer so verbessern können, dass er auch nicht lineare Klassifikationen vornehmen kann.

Kategoriale Eigenschaften

Die Vermittler-Datenmenge enthält sowohl numerische als auch kategoriale Daten. Manche Klassifizierer, wie zum Beispiel der Entscheidungsbaum, können mit beiden Arten umgehen, ohne die Daten aufbereiten zu müssen, aber die Klassifizierer im Rest dieses Kapitels funktionieren nur mit numerischen Daten. Um damit umgehen zu können, brauchen Sie eine Möglichkeit, Daten in Zahlen umzuwandeln, sodass sie zum Klassifizieren genutzt werden können.

Ja/Nein-Fragen

Am einfachsten lassen sich Ja/Nein-Fragen in Zahlen umwandeln, da Sie ein »Ja« in eine 1 und ein »Nein« in eine –1 konvertieren können. Damit haben Sie auch die Option, fehlende oder nicht eindeutige Daten (wie »Ich weiß nicht«) als 0 darzustellen. Fügen Sie die Funktion `yesno` der Datei *advancedclassify.py* hinzu, um diese Umwandlung für Sie vorzunehmen:

```
def yesno(v):
  if v=='yes': return 1
  elif v=='no': return -1
  else: return 0
```

e-bol.net

Interessenlisten

Es gibt ein paar unterschiedliche Möglichkeiten, die Interessen von Personen in der Datenmenge zu repräsentieren. Die einfachste ist, jedes mögliche Interesse als eigene numerische Variable darzustellen und eine 0 hinzuzufügen, wenn die Person dieses Interesse hat, bzw. 1, falls nicht. Haben Sie es mit einzelnen Personen zu tun, ist dies der beste Ansatz. In diesem Fall haben Sie aber Personenpaare, daher wäre ein intuitiveres Vorgehen, die Anzahl gemeinsamer Interessen als Variable zu nutzen.

Fügen Sie eine neue Funktion `matchcount` der Datei *advancedclassify.py* hinzu, die die Anzahl der übereinstimmenden Elemente in einer Liste als Gleitkommazahl zurückgibt:

```
def matchcount(interest1,interest2):
  l1=interest1.split(':')
  l2=interest2.split(':')
  x=0
  for v in l1:
    if v in l2: x+=1
  return x
```

Die Anzahl an gemeinsamen Interessen ist eine interessante Variable, aber damit verschwinden auch definitiv ein paar potenziell nützliche Informationen. Es ist möglich, dass bestimmte Kombinationen verschiedener Interessen gut zusammenpassen, wie zum Beispiel Skifahren und Snowboardfahren oder Trinken und Tanzen. Ein Klassifizierer, der nicht mit den ursprünglichen Daten trainiert wurde, wäre niemals in der Lage, diese Kombinationen zu erkennen.

Eine Alternative zum Erstellen einer neuen Variablen für jedes Interesse, das zu einer großen Menge von Variablen und damit zu deutlich komplexeren Klassifizierern führt, ist die Anordnung von Interessen in einer Hierarchie. Sie können zum Beispiel sagen, dass Ski- und Snowboardfahren beides Beispiele für Wintersportarten sind, was wiederum eine Unterkategorie von Sport ist. Ein Paar, das gemeinsam an Wintersport interessiert ist, wenn auch nicht am gleichen Sport, könnte zu ihrem `matchcount` eine 0,8 addiert bekommen statt eines ganzen Punkts. Je höher in der Hierarchie Sie gehen müssen, um eine Übereinstimmung zu finden, desto kleinere Bruchteile eines Punkts gäbe es. Auch wenn die Vermittler-Datenmenge diese Hierarchie nicht hat, ist dies eine Vorgehensweise, die Sie im Auge behalten können, wenn ähnliche Probleme auftauchen.

Entfernungen mithilfe von Yahoo! Maps bestimmen

Das größte Problem beim Umgang mit dieser Datenmenge ist der Standort. Es ist durchaus nachvollziehbar, dass Personen, die näher beieinanderleben, eher ein Paar ergeben, aber der Standort in der Datendatei wird als Kombination aus Adresse und

e-bol.net

Postleitzahl angegeben. Ein sehr einfaches Vorgehen wäre, »hat die gleiche PLZ« als Variable zu nutzen, aber das würde das Ganze doch sehr einschränken – es ist durchaus möglich, dass Personen in aneinandergrenzenden Straßen wohnen, aber unterschiedliche Postleitzahlen haben. Idealerweise würden Sie eine neue Variable definieren, die auf der Entfernung basiert.

Natürlich ist es nicht möglich, die Entfernung zwischen zwei Adressen ohne zusätzliche Informationen zu ermitteln. Glücklicherweise stellt Yahoo! Maps einen API-Service namens *Geocoding* bereit, der eine Adresse in den USA erwartet und deren Längen- und Breitenangabe zurückliefert. Mithilfe dieser Werte können Sie für ein Paar von Adressen eine ungefähre Entfernung zwischen ihnen ermitteln.

Wenn Sie aus irgendeinem Grund die Yahoo!-API nicht nutzen können, fügen Sie einfach eine Dummy-Funktion für `milesdistance` *advancedclassify.py* hinzu:

```
def milesdistance(a1,a2):
  return 0
```

Einen Yahoo! Application Key erhalten

Um die Yahoo!-API nutzen zu können, brauchen Sie als Erstes einen Application Key, der in Ihren Abfragen genutzt wird, um Ihre Anwendung zu identifizieren. Sie können einen Schlüssel bekommen, indem Sie die Seite *http://api.search.yahoo.com/webservices/register_application* aufrufen und ein paar Fragen beantworten. Sollten Sie noch kein Yahoo!-Konto haben, müssen Sie eines anlegen. Sie bekommen den Schlüssel direkt und müssen nicht einmal auf eine Antwort per E-Mail warten.

Verwenden der Geocoding-API

Die Geocoding-API erwartet einen Request an eine URL. Sie können dies unter *http://api.local.yahoo.com/MapsService/V1/geocode?appid=appid&location=location* vornehmen.

Der Wert von location ist Freitext und kann eine Adresse, eine Postleitzahl oder sogar eine Stadt und ein Bundesstaat sein. Das zurückgelieferte Ergebnis ist eine XML-Datei, die in etwa so aussieht:

```
<ResultSet>
<Result precision="address">
<Latitude>37.417312</Latitude>
<Longitude>-122.026419</Longitude>
<Address>755 FIRST AVE</Address>
<City>SUNNYVALE</City>
<State>CA</State>
<Zip>94089-1019</Zip>
<Country>US</Country>
</Result>
</ResultSet>
```

e-bol.net

Die für Sie interessanten Felder sind `Longitude` und `Latitude`. Um sie zu parsen, werden Sie die *minidom*-API verwenden, die Sie schon in früheren Kapiteln genutzt haben. Fügen Sie `getlocation` der Datei *advancedclassify.py* hinzu:

```
yahookey="Ihr Schlüssel"
from xml.dom.minidom import parseString
from urllib import urlopen,quote_plus

loc_cache={}
def getlocation(address):
  if address in loc_cache: return loc_cache[address]
  data=urlopen('http://api.local.yahoo.com/MapsService/V1/'+\
               'geocode?appid=%s&location=%s' %
               (yahookey,quote_plus(address))).read()
  doc=parseString(data)
  lat=doc.getElementsByTagName('Latitude')[0].firstChild.nodeValue
  long=doc.getElementsByTagName('Longitude')[0].firstChild.nodeValue
  loc_cache[address]=(float(lat),float(long))
  return loc_cache[address]
```

Diese Funktion erzeugt die URL mit Ihrem Anwendungsschlüssel und dem Standort und liefert Länge und Breite zurück. Auch wenn dies die einzigen Informationen sind, die Sie zum Berechnen der Entfernung benötigen, können Sie die Yahoo! Geocoding-API auch für andere Dinge nutzen, zum Beispiel zum Ermitteln der Postleitzahl für eine bestimmte Adresse oder zum Herausfinden, wo sich eine bestimmte Postleitzahl befindet.

Berechnen der Entfernung

Das Umwandeln der Längen und Breiten zweier Punkte in eine Entfernung (in Meilen) ist erstaunlicherweise ziemlich kompliziert, wenn Sie eine hohe Genauigkeit benötigen. Aber die Entfernungen sind in diesem Fall recht klein, und Sie berechnen sie ja sowieso nur zum Vergleichen, daher können Sie Näherungen vornehmen. Die Vereinfachung ähnelt dem euklidischen Abstand, den Sie aus den vorigen Kapiteln kennen, nur dass die Differenz zwischen den Breiten vorher mit 69,1 und die der Längen mit 53 multipliziert werden.

Fügen Sie die Funktion `milesdistance` der *advancedclassify.py* hinzu:

```
def milesdistance(a1,a2):
  lat1,long1=getlocation(a1)
  lat2,long2=getlocation(a2)
  latdif=69.1*(lat2-lat1)
  longdif=53.0*(long2-long1)
  return (latdif**2+longdif**2)**.5
```

Diese Funktion ruft die weiter oben definierte Funktion `getlocation` für beide Adressen auf und berechnet die Entfernung zwischen ihnen. Wenn Sie wollen, können Sie das in Ihrer Python-Session austesten:

e-bol.net

```
>>> reload(advancedclassify)
<module 'advancedclassify' from 'advancedclassify.py'>
>>> advancedclassify.getlocation('1 alewife center, cambridge, ma')
(42.398662999999999, -71.140512999999999)
>>> advancedclassify.milesdistance('cambridge, ma','new york,ny')
191.77952424273104
```

Die mithilfe der Näherung berechneten Entfernungen werden normalerweise weniger als 10 % Fehler enthalten, was für diese Anwendung ausreicht.

Erzeugen der neuen Datenmenge

Sie haben nun alle erforderlichen Elemente beisammen, um eine Datenmenge zum Trainieren des Klassifizierers zu erstellen. Sie brauchen jetzt eine Funktion, die alles kombiniert. Diese Funktion wird die Datenmenge mithilfe der Funktion `loadmatch` aus der Datei laden und die entsprechenden Transformationen auf die Spalten anwenden. Fügen Sie `loadnumerical` der Datei *advancedclassify.py* hinzu:

```
def loadnumerical():
  oldrows=loadmatch('matchmaker.csv')
  newrows=[]
  for row in oldrows:
    d=row.data
    data=[float(d[0]),yesno(d[1]),yesno(d[2]),
          float(d[5]),yesno(d[6]),yesno(d[7]),
          matchcount(d[3],d[8]),
          milesdistance(d[4],d[9]),
          row.match]
    newrows.append(matchrow(data))
  return newrows
```

Diese Funktion erzeugt eine neue Datenzeile für jede Zeile in der ursprünglichen Menge. Sie ruft die weiter oben definierten Funktionen zum Umwandeln aller Daten in Zahlen auf, einschließlich der Entfernungsberechnung und dem Zähler für gemeinsame Interessen.

Rufen Sie diese Funktion aus Ihrer Python-Session auf, um die neue Datenmenge zu erstellen:

```
>>> reload(advancedclassify)
>>> numericalset=advancedclassify.loadnumerical()
>>> numericalset[0].data
[39.0, 1, -1, 43.0, -1, 1, 0, 0.90110601059793416]
```

Auch hier ist es wieder sehr einfach, Untermengen zu erzeugen, indem Sie die für Sie interessanten Spalten angeben. Das ist nützlich, wenn es um das Visualisieren der Daten und um ein Verständnis der Funktion des Klassifizierers für verschiedene Variablen geht.

e-bol.net

Skalieren der Daten

Solange Sie nur Vergleiche anhand des Alters von Personen vorgenommen haben, war es kein Problem, die Daten so, wie sie waren, zu nutzen, um Durchschnitts- und Abstandswerte zu ermitteln – denn es ist sinnvoll, Variablen zu vergleichen, die das Gleiche bedeuten. Aber nun haben Sie ein paar neue Variablen eingeführt, die nicht mit dem Alter vergleichbar sind, da ihre Werte in einem deutlich niedrigeren Bereich liegen. Eine unterschiedliche Meinung zu Kindern – ein Abstand von 2, zwischen 1 und –1 – kann in Wirklichkeit deutlich entscheidender sein als eine Altersdifferenz von sechs Jahren, aber wenn Sie die Daten so verwenden, wie Sie sie erhalten haben, würde die Altersdifferenz dreimal so viel zählen.

Um dieses Problem zu lösen, ist es keine schlechte Idee, alle Daten auf eine gemeinsame Skala zu bringen, sodass sich die Abstände von jeder Variablen vergleichen lassen. Sie können dies tun, indem Sie die niedrigsten und höchsten Werte für jede Variable bestimmen und die Daten so skalieren, dass der niedrigste Wert nun 0 und der höchste Wert 1 ist, während sich alle anderen Werte irgendwo dazwischen ansiedeln.

Fügen Sie `scaledata` der Datei *advancedclassifier.py* hinzu:

```
def scaledata(rows):
  low=[999999999.0]*len(rows[0].data)
  high=[-999999999.0]*len(rows[0].data)
  # Niedrigste und höchste Werte finden.
  for row in rows:
    d=row.data
    for i in range(len(d)):
      if d[i]<low[i]: low[i]=d[i]
      if d[i]>high[i]: high[i]=d[i]

  # Eine Funktion erstellen, die die Daten skaliert.
  def scaleinput(d):
    return [(d.data[i]-low[i])/(high[i]-low[i])
            for i in range(len(low))]

  # Alle Daten skalieren.
  newrows=[matchrow(scaleinput(row.data)+[row.match])
           for row in rows]

  # Neue Daten und die Funktion zurückgeben.
  return newrows,scaleinput
```

Diese Funktion definiert eine interne Funktion `scaleinput`, die den niedrigsten Wert nutzt und ihn von allen anderen Werten abzieht, um den Bereich bei 0 beginnen zu lassen. Dann werden die Werte durch die Differenz zwischen niedrigstem und höchstem Wert geteilt, um alle auf den Bereich zwischen 0 und 1 zu skalieren. Die Funktion wendet `scaleinput` auf jede Zeile in der Datenmenge an und liefert die

e-bol.net

neue Datenmenge zusammen mit der Funktion zurück, sodass Sie auch Ihre Abfragen skalieren können.

Jetzt können Sie den linearen Klassifizierer mit einer größeren Anzahl an Variablen ausprobieren:

```
>>> reload(advancedclassify)
<module 'advancedclassify' from 'advancedclassify.py'>
>>> scaledset,scalef=advancedclassify.scaledata(numericalset)
>>> avgs=advancedclassify.lineartrain(scaledset)
>>> numericalset[0].data
[39.0, 1, -1, 43.0, -1, 1, 0, 0.90110601059793416]
>>> numericalset[0].match
0
>>> advancedclassify.dpclassify(scalef(numericalset[0].data),avgs)
1
>>> numericalset[11].match
1
>>> advancedclassify.dpclassify(scalef(numericalset[11].data),avgs)
1
```

Beachten Sie, dass Sie die numerischen Beispiele zuerst skalieren müssen, damit sie in den neuen Raum passen. Auch wenn der Klassifizierer für ein paar Beispiele funktioniert, werden die Einschränkungen durch den Versuch, einfach eine Trennlinie zu finden, nun deutlicher. Um die Ergebnisse zu verbessern, müssen Sie nun über die lineare Klassifikation hinausgehen.

Kernel-Methoden verstehen

Überlegen Sie, was passiert wäre, wenn Sie versucht hätten, den linearen Klassifizierer auf eine Datenmenge anzuwenden, die so aussieht wie die in Abbildung 9-7.

Wo würden die Mittelpunkte für jede Klasse liegen? Genau am selben Ort! Auch wenn Ihnen und mir klar ist, dass alles innerhalb des Kreises ein X und alles außerhalb ein O ist, kann der lineare Klassifizierer nicht zwischen diesen beiden Klassen unterscheiden.

Aber stellen Sie sich vor, was passierte, wenn Sie jeden x- und y-Wert zuerst quadrierten. Ein Punkt, der sich an (–1;2) befand, würde nun bei (1;4) liegen, ein Punkt bei (0,5;1) käme nun zu (0,25;1) und so weiter. Der neue Graph würde aussehen wie der in Abbildung 9-8.

Alle X wurden in die Ecke verschoben und alle O liegen außerhalb. Es ist nun sehr einfach, die X und O durch eine gerade Linie zu trennen, und immer wenn ein neues Element zu klassifizieren ist, brauchen Sie nur die x- und y-Werte zu quadrieren und zu schauen, auf welcher Seite der Linie der Punkt landet.

Dieses Beispiel zeigt, dass es durch das Transformieren der Punkte möglich ist, eine neue Datenmenge zu erstellen, dass sich durch eine gerade Linie trennen lässt.

e-bol.net

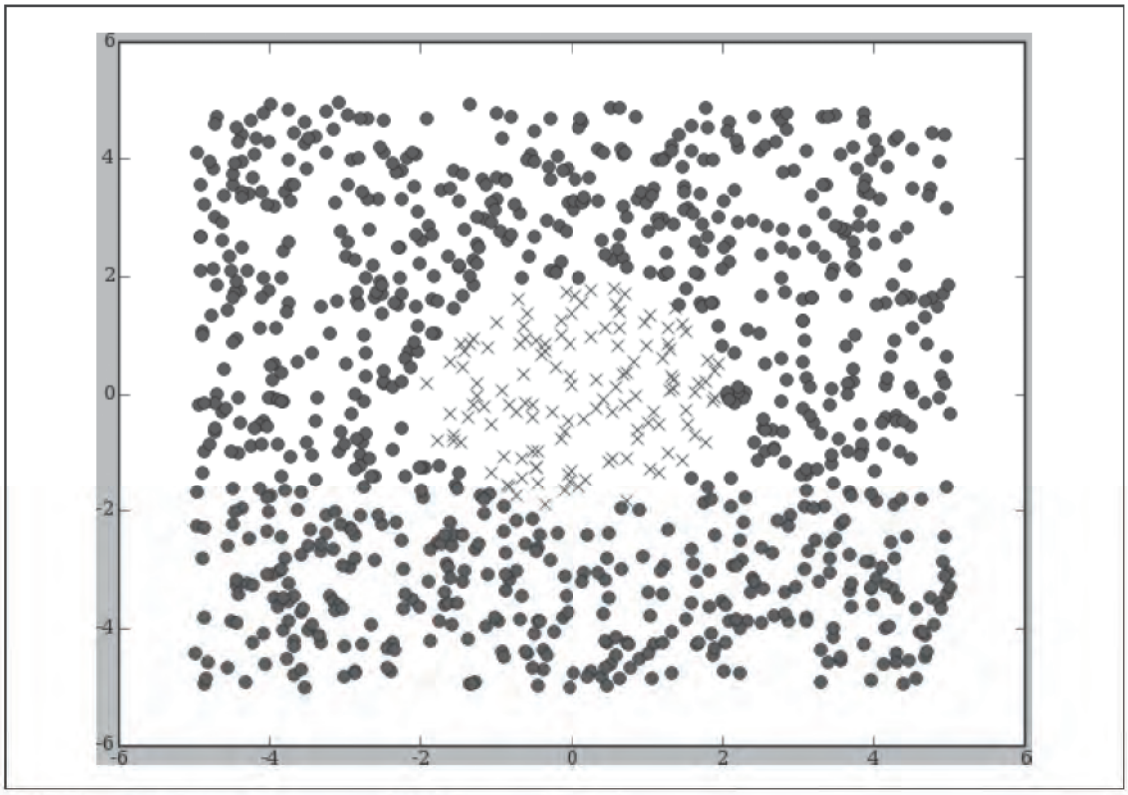

Abbildung 9-7: Eine Klasse umschließt eine andere Klasse

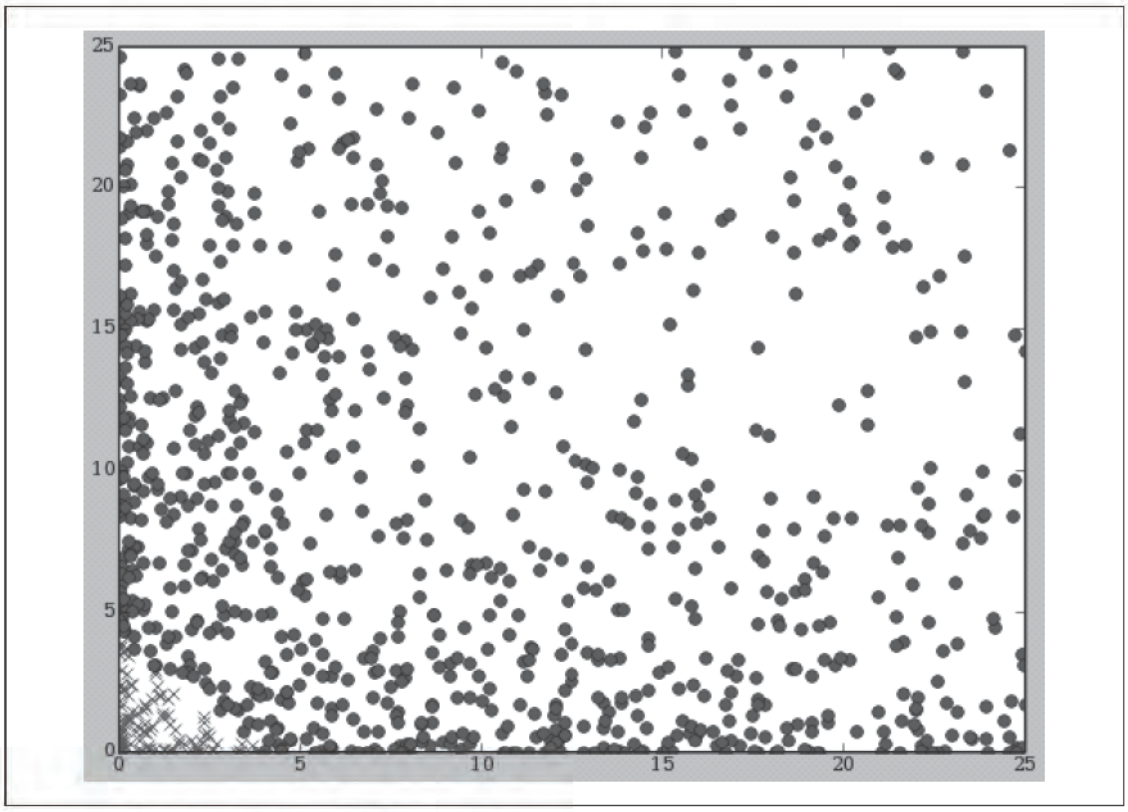

Abbildung 9-8: Verschieben der Punkte in einen anderen Raum

e-bol.net

Allerdings wurde dieses Beispiel gewählt, weil es sich eben so leicht umwandeln lässt – bei echten Problemen wird die Transformation sehr wahrscheinlich deutlich komplizierter sein, und die Daten werden in mehr Dimensionen umgewandelt werden müssen. So müssen Sie zum Beispiel eventuell eine Datenmenge mit x- und y-Werten umwandeln, sodass eine neue Datenmenge mit a-, b- und c-Koordinaten entsteht, wobei gilt: $a = x^2$, $b = x * y$ und $c = y^2$. Nachdem die Daten in mehr Dimensionen gebracht wurden, ist es einfacher, die Trennlinie zwischen zwei Klassen zu finden.

Der Kernel-Trick

Sie könnten zwar Code schreiben, der die Daten in einen neuen Raum bringt, dies wird in der Praxis aber selten gemacht, da das Finden einer Trennlinie bei der Arbeit mit echten Daten es erforderlich machen kann, die Daten in Hunderte oder Tausende von Dimensionen zu transformieren, was sich wirklich schlecht implementieren lässt. Aber bei jedem Algorithmus, der Skalarprodukte nutzt – einschließlich des linearen Klassifizierers –, können Sie eine Technik nutzen, die als *Kernel-Trick* bezeichnet wird.

Beim Kernel-Trick geht es darum, die Skalarproduktfunktion durch eine neue Funktion zu ersetzen, die das zurückliefert, was das Skalarprodukt zurückliefern würde, wenn die Daten vorher mit einer Abbildungsfunktion in einen höherdimensionalen Raum transformiert worden wären. Dabei gibt es keine Beschränkung der möglichen Transformationen, allerdings werden in der Praxis nur ein paar verwendet. Die normalerweise empfohlene (die Sie auch hier nutzen werden) wird als *Radial-Basisfunktion* (RBF) bezeichnet.

Die Radial-Basisfunktion verhält sich wie das Skalarprodukt: Sie erwartet zwei Vektoren und liefert einen (Skalar-)Wert zurück. Aber anders als das Skalarprodukt ist sie nicht linear und kann daher komplexere Räume abbilden. Fügen Sie `rbf` *advancedclassify.py* hinzu:

```
def rbf(v1,v2,gamma=20):
  dv=[v1[i]-v2[i] for i in range(len(v1))]
  l=veclength(dv)
  return math.e**(-gamma*l)
```

Diese Funktion hat auch noch einen weiteren Parameter `gamma`, der angepasst werden kann, um für eine gegebene Datenmenge die beste lineare Aufteilung zu erreichen.

Sie brauchen nun eine neue Funktion, die die Abstände zu den Mittelpunkten im transformierten Raum berechnet. Leider wurden die Mittelwerte im Ursprungsraum ermittelt, daher können sie hier nicht verwendet werden – tatsächlich können die Mittelwerte gar nicht berechnet werden, da Sie die Position der Punkte im neuen Raum nicht ermitteln werden. Glücklicherweise ergibt eine Mittelung über

e-bol.net

eine Reihe von Vektoren und das Bilden des Skalarprodukts des Mittelwerts mit Vektor A das gleiche Ergebnis wie das Mitteln der Skalarprodukte von Vektor A mit jedem Vektor in der Menge.

Anstatt also das Skalarprodukt zwischen dem Punkt, den Sie klassifizieren wollen, und dem Mittelpunkt einer Klasse zu bilden, berechnen Sie das Skalarprodukt oder die Radial-Basisfunktion zwischen dem Punkt und jedem anderen Punkt in der Klasse und bilden dann den Mittelwert. Fügen Sie `nlclassify` der Datei *advancedclassify.py* hinzu:

```
def nlclassify(point,rows,offset,gamma=10):
  sum0=0.0
  sum1=0.0
  count0=0
  count1=0

  for row in rows:
    if row.match==0:
      sum0+=rbf(point,row.data,gamma)
      count0+=1
    else:
      sum1+=rbf(point,row.data,gamma)
      count1+=1
  y=(1.0/count0)*sum0-(1.0/count1)*sum1+offset

  if y<0: return 0
  else: return 1

def getoffset(rows,gamma=10):
  l0=[]
  l1=[]
  for row in rows:
    if row.match==0: l0.append(row.data)
    else: l1.append(row.data)
  sum0=sum(sum([rbf(v1,v2,gamma) for v1 in l0]) for v2 in l0)
  sum1=sum(sum([rbf(v1,v2,gamma) for v1 in l1]) for v2 in l1)

  return (1.0/(len(l1)**2))*sum1-(1.0/(len(l0)**2))*sum0
```

Der Offset-Wert ist im transformierten Raum ebenfalls anders, wobei das Berechnen ein wenig langsam sein kann. Aus diesem Grund sollten Sie ihn einmalig für eine Datenmenge berechnen und dann jedes Mal an `nlclassify` übergeben.

Probieren Sie diesen neuen Klassifizierer nur mit den Alterswerten aus und prüfen Sie, ob er die Probleme behebt, die Sie entdeckt haben:

```
>>> advancedclassify.nlclassify([30,30],agesonly,offset)
1
>>> advancedclassify.nlclassify([30,25],agesonly,offset)
1
>>> advancedclassify.nlclassify([25,40],agesonly,offset)
0
```

e-bol.net

```
>>> advancedclassify.nlclassify([48,20],agesonly,offset)
0
```

Ausgezeichnet! Die Transformation ermöglicht es dem Klassifizierer zu erkennen, dass es einen Streifen mit zueinanderpassenden Kombinationen gibt, bei denen die Alterswerte nahe beieinanderliegen, während sich auf den Seiten des Streifen Bereiche befinden, in denen die Personen eher nicht zueinanderpassen. Jetzt erkennt der Klassifizierer, dass 48 im Verhältnis zu 20 keine gute Kombination ist. Probieren Sie es noch einmal mit den weiteren Daten aus:

```
>>> ssoffset=advancedclassify.getoffset(scaledset)
>>> numericalset[0].match
0
>>> advancedclassify.nlclassify(scalef(numericalset[0].data),scaledset,ssoffset)
0
>>> numericalset[1].match
1
>>> advancedclassify.nlclassify(scalef(numericalset[1].data),scaledset,ssoffset)
1
>>> numericalset[2].match
0
>>> advancedclassify.nlclassify(scalef(numericalset[2].data),scaledset,ssoffset)
0
>>> newrow=[28.0,-1,-1,26.0,-1,1,2,0.8] # Mann will keine Kinder, Frau schon
>>> advancedclassify.nlclassify(scalef(newrow),scaledset,ssoffset)
0
>>> newrow=[28.0,-1,1,26.0,-1,1,2,0.8] # Beide wollen Kinder
>>> advancedclassify.nlclassify(scalef(newrow),scaledset,ssoffset)
1
```

Die Leistungsfähigkeit des Klassifizierers hat sich deutlich verbessert. In der obigen Session können Sie feststellen, dass ein Paar nicht zusammenkommen wird, wenn der Mann keine Kinder will, die Frau aber schon, selbst wenn sie vom Alter her nahe beieinanderliegen und zwei gemeinsame Interessen haben. Änder Sie versuchsweise auch die anderen Variablen einmal, um herauszufinden, wie das Ergebnis davon beeinflusst wird.

Support-Vektor-Maschinen

Denken Sie nochmals an die Aufgabe, eine gerade Linie zu finden, die zwei Klassen unterteilt. Abbildung 9-9 zeigt ein Beispiel. Die Durchschnittswerte jeder Klasse sind in der Abbildung ebenso zu sehen wie die Trennlinie, die durch sie entsteht.

Beachten Sie, dass die Trennlinie, die aufgrund der Durchschnittswerte berechnet wurde, zwei der Punkte falsch klassifiziert, weil sie viel näher an der Linie liegen als der Rest der Daten. Das Problem ist, dass die meisten Daten weit entfernt von der Linie liegen und die beiden Punkte bei der Berechnung der Linie daher nicht relevant sind.

e-bol.net

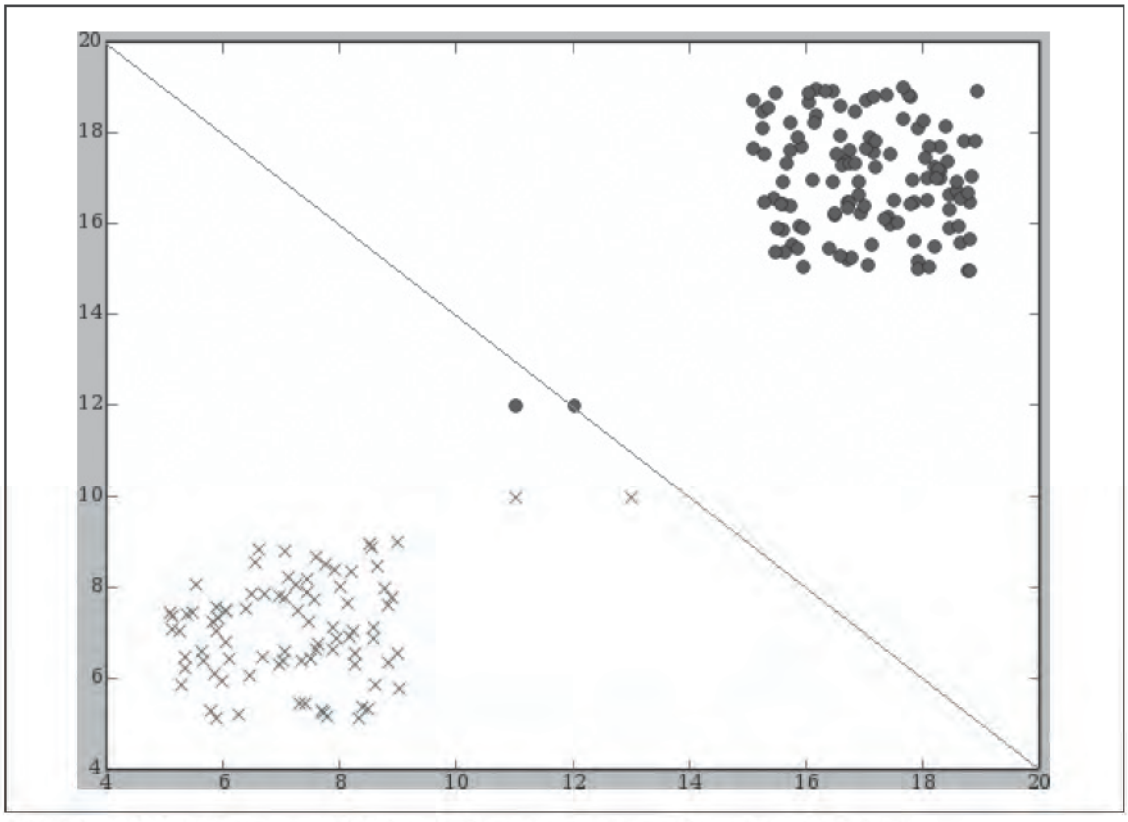

Abbildung 9-9: Der lineare Mittelwert-Klassifizierer klassifiziert Punkte falsch

Support-Vektor-Maschinen sind ein bekanntes Set von Methoden, um Klassifizierer zu erzeugen, die dieses Problem lösen. Das tun sie, indem sie versuchen, die Linie zu finden, die von jeder Klasse so weit wie möglich entfernt ist. Diese Linie wird als *maximal trennende Hyperebene* (*Maximum-Margin Hyperplane*) bezeichnet und ist in Abbildung 9-10 zu sehen.

Die Trennlinie wurde so gewählt, dass die parallelen Linien, die die Elemente jeder Klasse berühren, so weit wie möglich von ihr entfernt sind. Auch hier können Sie bestimmen, in welche Klasse ein neuer Datenpunkt gehört, indem Sie einfach schauen, auf welcher Seite der Linie er sich befindet. Beachten Sie, dass nur die Punkte am »Rand« benötigt werden, um die Position der Trennlinie zu bestimmen – Sie könnten alle anderen Daten verwerfen, und die Linie würde trotzdem an der gleichen Stelle liegen. Die Punkte nahe der Linie werden als *Stützvektoren* (*Support Vectors*) bezeichnet. Der Algorithmus, der die Stützvektoren findet und nutzt, um die Trennlinie zu ermitteln, ist die Support-Vektor-Maschine.

Sie haben schon gesehen, wie ein linearer Klassifizierer in einen nicht linearen Klassifizierer mithilfe des Kernel-Tricks umgewandelt werden kann, solange das Skalarprodukt zum Vergleich genutzt wird. Support-Vektor-Maschinen verwenden ebenfalls das Skalarprodukt, daher können sie mit Kerneln genutzt werden, um nicht lineare Klassifikationen vorzunehmen.

e-bol.net

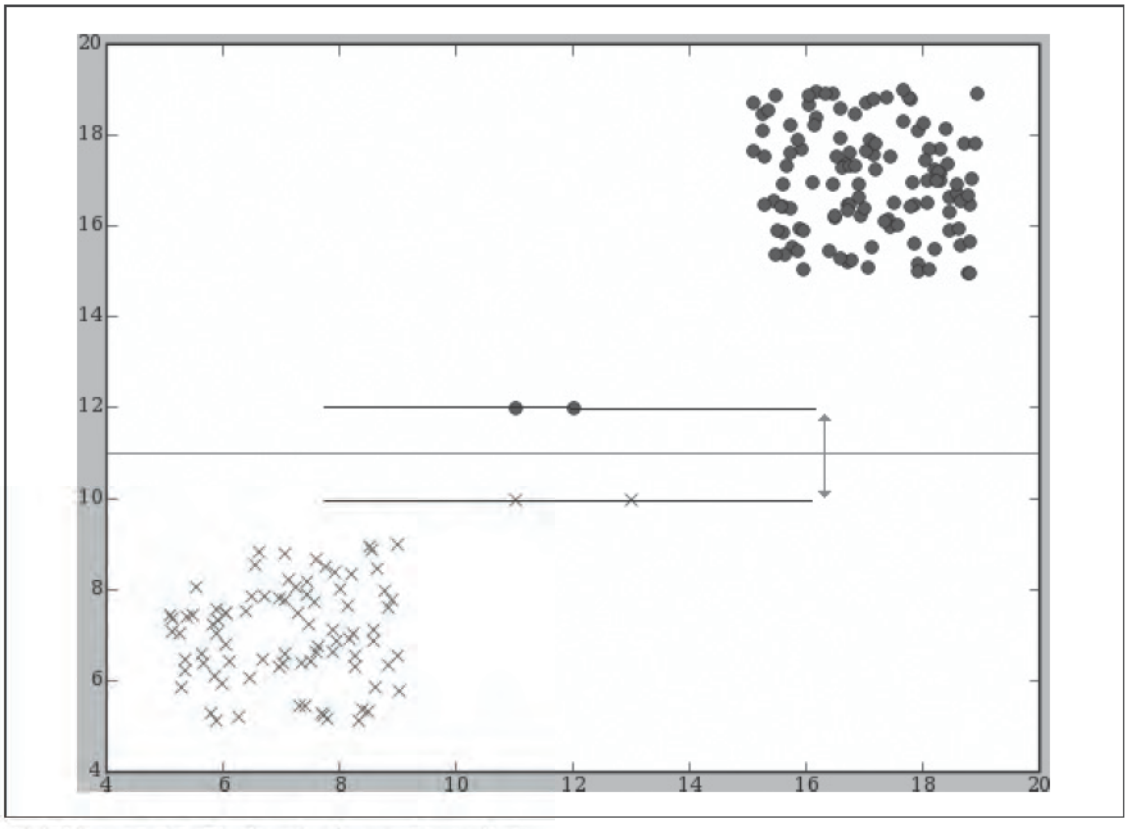

Abbildung 9-10: Finden der besten Trennlinie

Anwendungen von Support-Vektor-Maschinen

Da Support-Vektor-Maschinen gut mit mehrdimensionalen Datenmengen umgehen können, werden sie am häufigsten auf datenintensive wissenschaftliche und andere Probleme angewandt, die mit sehr komplexen Datenmengen zu tun haben. Dazu gehören zum Beispiel:

- Klassifikation von Gesichtsausdrücken
- Erkennen von Eindringlingen im Militärbereich
- Vorhersagen von Proteinstrukturen aus ihren Sequenzen
- Handschriftenerkennung
- Bestimmung des Schadenspotenzials bei Erdbeben

Verwenden der LIBSVM

Die Erläuterung im vorigen Abschnitt sollte Ihnen dabei helfen zu verstehen, wie und warum Support-Vektor-Maschinen funktionieren, aber der Algorithmus zum Trainieren einer Support-Vektor-Maschine beinhaltet mathematische Konzepte, die

e-bol.net

sehr rechenintensiv sind und auch nicht in den Rahmen dieses Kapitels passen. Aus diesem Grund wird dieser Abschnitt eine Open Source-Bibliothek namens *LIBSVM* vorstellen, die ein SVM-Modell trainieren, Vorhersagen treffen und die Vorhersagen innerhalb eines Datenmodells testen kann. Sie unterstützt sogar die Radial-Basisfunktion und andere Kernel-Methoden.

Beschaffen der LIBSVM

Sie können LIBSVM von *http://www.csie.ntu.edu.tw/~cjlin/libsvm* herunterladen.

LIBSVM ist in C++ geschrieben und enthält zudem eine Version in Java. Das herunterzuladende Paket enthält auch einen Python-Wrapper namens *svm.py*. Um diesen nutzen zu können, benötigen Sie eine kompilierte Version der LIBSVM für Ihre Plattform. Für Windows gibt es in dem Paket schon eine DLL namens *svmc.dll*. (Bei Python 2.5 müssen Sie diese Datei umbenennen in *svmc.pyd*, da es keine Bibliotheken mit DLL-Erweiterung importieren kann.) Die Dokumentation von LIBSVM erläutert, wie die Bibliothek für andere Plattformen kompiliert werden kann.

Eine Beispiel-Session

Nachdem Sie eine kompilierte Version der LIBSVM haben, kopieren Sie diese und *svm.py* in Ihren Python-Pfad oder Ihr Arbeitsverzeichnis. Sie können nun die Bibliothek in Ihrer Python-Session importieren und an einem einfachen Problem ausprobieren:

```
>>> from svm import *
```

Als Erstes erstellen Sie eine einfache Datenmenge. LIBSVM liest die Daten aus einem Tupel mit zwei Listen ein. Die erste Liste enthält die Klassen, die zweite die Eingangsdaten. Versuchen Sie, eine Datenmenge mit zwei möglichen Klassen zu erstellen:

```
>>> prob = svm_problem([1,-1],[[1,0,1],[-1,0,-1]])
```

Außerdem müssen Sie per `svm_parameter` festlegen, welchen Kernel Sie nutzen wollen:

```
>>> param = svm_parameter(kernel_type = LINEAR, C = 10)
```

Als Nächstes können Sie das Modell trainieren:

```
>>> m = svm_model(prob, param)
*
optimization finished, #iter = 1
nu = 0.025000
obj = -0.250000, rho = 0.000000
nSV = 2, nBSV = 0
Total nSV = 2
```

e-bol.net

Und schließlich können Sie dieses Modell nutzen, um Vorhersagen über neue Elemente zu treffen:

```
>>> m.predict([1, 1, 1])
1.0
```

Damit haben Sie die gesamte Funktionalität der LIBSVM kennengelernt, die Sie benötigen, um ein Modell aus Trainingsdaten zu erstellen und Vorhersagen zu treffen. LIBSVM enthält noch ein weiteres nettes Feature – die Fähigkeit, die von Ihnen erstellten Trainingsmodelle zu speichern und wieder zu laden:

```
>>> m.save(test.model)
>>> m=svm_model(test.model)
```

Anwenden der SVM auf die Vermittler-Datenmenge

Um LIBSVM mit der Vermittler-Datenmenge zu nutzen, müssen Sie sie in das Listen-Tupel umwandeln, das von `svm_model` benötigt wird. Das ist eine einfache Transformation aus `scaledset` – Sie können es direkt in Ihrer Python-Session umsetzen:

```
>>> answers,inputs=[r.match for r in scaledset],[r.data for r in scaledset]
```

Erneut nutzen wir die skalierten Daten, um eine Übergewichtung bestimmter Variablen zu vermeiden, wodurch die Leistungsfähigkeit des Algorithmus verbessert wird. Verwenden Sie diese neue Funktion, um die neue Datenmenge zu erstellen und ein Modell mit der Radial-Basisfunktion als Kernel aufzubauen:

```
>>> param = svm_parameter(kernel_type = RBF)
>>> prob = svm_problem(answers,inputs)
>>> m=svm_model(prob,param)
*
optimization finished, #iter = 319
nu = 0.777538
obj = -289.477708, rho = -0.853058
nSV = 396, nBSV = 380
Total nSV = 396
```

Nun können Sie Vorhersagen darüber treffen, ob Personen mit gegebenen Attributkombinationen zusammenpassen werden oder nicht. Sie müssen die Funktion `scale` nutzen, um die Daten zu skalieren, für die Sie Vorhersagen haben wollen, sodass sich die Variablen im selben Wertebereich befinden wie die, mit denen Sie das Modell aufgebaut haben:

```
>>> newrow=[28.0,-1,-1,26.0,-1,1,2,0.8] # Mann will keine Kinder, Frau schon.
>>> m.predict(scalef(newrow))
0.0
>>> newrow=[28.0,-1,1,26.0,-1,1,2,0.8] # Beide wollen Kinder.
>>> m.predict(scalef(newrow))
1.0
```

e-bol.net

Auch wenn es so aussieht, als sei die Vorhersage gut, wäre es schön zu wissen, wie gut sie ist, sodass Sie die besten Parameter für Ihre Basisfunktion wählen können. LIBSVM enthält auch die Funktionalität, um die Modelle per Kreuzvalidierung zu überprüfen. Sie haben die Kreuzvalidierung bereits in Kapitel 8 kennengelernt – die Datenmenge wird automatisch in Trainings- und Testsets aufgeteilt. Die Trainingssets werden genutzt, um das Modell aufzubauen, während das Modell mit den Testsets geprüft wird, um zu sehen, wie gut die Vorhersagen sind.

Sie können die Qualität des Modells mit der Kreuzvalidierungsfunktion überprüfen. Diese Funktion erwartet einen Parameter `n` und teilt die Datenmenge in *n* Untermengen auf. Dann nutzt sie jede Untermenge als Testset und trainiert das Modell mit allen anderen Untermengen. Sie liefert eine Liste mit Ergebnissen zurück, die Sie mit der ursprünglichen Liste vergleichen können.

```
>>> guesses = cross_validation(prob, param, 4)
...
>>> guesses
[0.0, 0.0, 0.0, 0.0, 1.0, 0.0,...
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,...
...]
>>> sum([abs(answers[i]-guesses[i]) for i in range(len(guesses))])
116.0
```

Die Summe der Unterschiede zwischen den Antworten und Schätzungen ist 116. Da es in der ursprünglichen Datenmenge 500 Zeilen gab, bedeutet dies, dass der Algorithmus 384 Übereinstimmungen hatte. Wenn Sie möchten, können Sie sich andere Kernel und Parameter in der Dokumentation der LIBSVM anschauen, um zu versuchen, durch Ändern von `param` Verbesserungen zu erreichen.

Vermitteln bei Facebook

Facebook ist eine beliebte Social Networking Site, die ursprünglich für Studenten gedacht war, sich aber mittlerweile einem größeren Publikum geöffnet hat. Wie andere Social Networking Sites ermöglicht Facebook den Benutzern, Profile zu erstellen, demografische Daten über sich selbst einzutragen und Verbindungen zu ihren Freunden auf der Site herzustellen. Facebook enthält zudem eine API, mit der Sie Informationen über Personen abrufen und herausfinden können, ob zwei Benutzer Freunde sind oder nicht. Dadurch können Sie eine ähnliche Datenmenge wie die Kuppel-Datenmenge erstellen, die aber Daten von echten Personen nutzt.

Während dieses Buch entsteht, fühlt sich Facebook immer noch stark dem Schutz der Privatsphäre ihrer Benutzer verpflichtet, daher können Sie nur die Profile von Personen anschauen, die Ihre Freunde sind. Für die API gelten die gleichen Regeln – man muss sich anmelden und kann nur eingeschränkte Abfragen durchführen – daher können Sie diesen Abschnitt nur mit eigenen Daten nachvollziehen, wenn Sie

e-bol.net

ein Konto bei Facebook haben und dort mit mindestens 20 Personen verbunden sind.

Einen Entwicklerschlüssel erhalten

Wenn Sie ein Konto bei Facebook haben, können Sie sich auf der Facebook-Entwicklersite einen Entwicklerschlüssel organisieren. Sie erhalten ihn unter *http://developers.facebook.com*.

Sie werden zwei Strings bekommen – einen API-Schlüssel und einen »geheimen« Schlüssel. Der API-Schlüssel dient dazu, Sie zu identifizieren, während der geheime Schlüssel genutzt wird, um Ihre Anfragen in der Hash-Funktion zu verschlüsseln, die Sie später noch sehen werden. Zu Beginn legen Sie erst einmal eine neue Datei namens *facebook.py* an, importieren ein paar Module, die Sie benötigen, und legen Konstanten fest:

```
import urllib,md5,webbrowser,time
from xml.dom.minidom import parseString

apikey="Ihr API-Schlüssel"
secret="Ihr geheimer Schlüssel"
FacebookSecureURL = "https://api.facebook.com/restserver.php"
```

Es gibt noch zwei Hilfsfunktionen, die Sie hinzufügen sollten: `getsinglevalue` liefert den nächsten Wert innerhalb eines angegebenen Knotens, während `callid` eine Zahl zurückliefert, die auf der Systemzeit basiert.

```
def getsinglevalue(node,tag):
  nl=node.getElementsByTagName(tag)
  if len(nl)>0:
    tagNode=nl[0]
    if tagNode.hasChildNodes( ):
      return tagNode.firstChild.nodeValue
  return ''

def callid( ):
  return str(int(time.time( )*10))
```

Einige der Aufrufe bei Facebook erfordern das Übermitteln einer Sequenznummer, die beliebig sein kann, solange sie größer als die letzte verwendete Sequenznummer ist. Durch die Verwendung der Systemzeit erhalten Sie eine einfache Möglichkeit, konsistent größer werdende Zahlen zu bekommen.

Erstellen einer Session

Die Prozedur zum Erstellen einer Session bei Facebook ist darauf ausgelegt, dass Sie eine Anwendung für andere Personen erstellen können, ohne etwas über ihre Login-Informationen zu erfahren. Dazu benötigen Sie mehrere Schritte:

e-bol.net

1. Nutzen Sie die Facebook-API, um ein Token anzufordern.
2. Schicken Sie dem Benutzer eine URL für die Login-Seite von Facebook, wobei die URL das Token enthält.
3. Warten Sie, bis sich der Benutzer angemeldet hat.
4. Fordern Sie von Facebook mithilfe des Tokens eine Session an.

Da während all dieser Aufrufe viele Variablen genutzt werden, ist es besser, die Facebook-Funktionalität in einer Klasse unterzubringen. Erstellen Sie eine neue Klasse namens `fbsession` in *facebook.py* und fügen Sie eine Methode `__init__` hinzu, die die obigen Schritte umsetzt:

```
class fbsession:
  def __init__(self):
    self.session_secret=None
    self.session_key=None
    self.token=self.createtoken( )
    webbrowser.open(self.getlogin( ))
    print "Drücken Sie nach dem Anmelden die Return-Taste:",
    raw_input( )
    self.getsession( )
```

Es gibt eine Reihe von Methoden, die von `__init__` genutzt werden und der Klasse hinzuzufügen sind. Als Erstes brauchen Sie eine Möglichkeit, Requests an die Facebook-API zu schicken. Die Methode `sendrequest` öffnet eine Verbindung zu Facebook und postet die Argumente. Eine XML-Datei wird zurückgegeben und mit dem *minidom*-Parser analysiert. Fügen Sie diese Methode Ihrer Klasse hinzu:

```
def sendrequest(self, args):
  args['api_key'] = apikey
  args['sig'] = self.makehash(args)
  post_data = urllib.urlencode(args)
  url = FacebookURL + "?" + post_data
  data=urllib.urlopen(url).read( )
  return parseString(data)
```

Die fett gedruckte Zeile erzeugt die Signatur des Requests. Dazu wird die Methode `makehash` genutzt, die alle Argumente aneinanderhängt und mit dem geheimen Schlüssel hasht. Sie werden gleich sehen, dass sich der geheime Schlüssel ändert, sobald Sie eine Session haben, daher prüft diese Methode vorher, ob Sie schon einen geheimen Session-Schlüssel besitzen. Fügen Sie `makehash` Ihrer Klasse hinzu:

```
def makehash(self,args):
  hasher = md5.new(''.join([x + '=' + args[x] for x in sorted(args.keys( ))]))
  if self.session_secret: hasher.update(self.session_secret)
  else: hasher.update(secret)
  return hasher.hexdigest( )
```

Nun können Sie Anfragen an die Facebook-API stellen. Beginnen wir mit `createtoken`, das ein Token erstellt und abspeichert, um es auf der Login-Seite zu nutzen:

e-bol.net

```
def createtoken(self):
  res = self.sendrequest({'method':"facebook.auth.createToken"})
  self.token = getsinglevalue(res,'token')
```

Fügen Sie auch noch `getlogin` hinzu, das einfach die URL für die Anmeldeseite zurückgibt:

```
def getlogin(self):
  return "http://api.facebook.com/login.php?api_key="+apikey+\
         "&auth_token=" + self.token
```

Nachdem sich der Benutzer angemeldet hat, sollte `getsession` aufgerufen werden, um einen Session-Schlüssel und den geheimen Session-Schlüssel zu erhalten, der für das Hashen zukünftiger Requests benötigt wird. Fügen Sie diese Methode Ihrer Klasse hinzu:

```
def getsession(self):
  doc=self.sendrequest({'method':'facebook.auth.getSession',
                        'auth_token':self.token})
  self.session_key=getsinglevalue(doc,'session_key')
  self.session_secret=getsinglevalue(doc,'secret')
```

Es ist ziemlich aufwendig, eine Facebook-Session einzurichten, aber wenn man den Prozess erst einmal durchlaufen hat, ist der Rest ziemlich einfach. Dieses Kapitel zeigt nur, wie man Informationen über Personen erhält, aber wenn Sie sich die Dokumentation anschauen, werden Sie feststellen, wie Sie ganz einfach Methoden zum Herunterladen von Bildern und Events aufrufen können.

Daten von Freunden herunterladen

Nun können Sie endlich damit beginnen, wirklich nützliche Funktionen anzulegen. Die Methode `getfriends` lädt die Liste der Freunde-IDs des momentan angemeldeten Benutzers herunter und liefert sie als Liste zurück. Fügen Sie diese Methode `fbsession` hinzu:

```
def getfriends(self):
  doc=self.sendrequest({'method':'facebook.friends.get',
                        'session_key':self.session_key,'call_id':callid()})
  results=[]
  for n in doc.getElementsByTagName('result_elt'):
    results.append(n.firstChild.nodeValue)
  return results
```

Da `getfriends` nur IDs zurückgibt, benötigen Sie eine weitere Methode, um auch echte Informationen über diese Personen zu erhalten. Die Methode `getinfo` ruft die API-Funktion `getInfo` mit einer Liste von Benutzer-IDs auf. Sie fragt eine nur kleine Zahl von ausgewählten Feldern ab, aber Sie können diese Liste erweitern, indem Sie `fields` noch mehr Felder hinzufügen und den Parsing-Code anpassen, damit dieser die relevanten Informationen ermittelt. Eine vollständige Liste mit Feldern findet sich in der Entwickler-Dokumentation von Facebook.

e-bol.net

```
def getinfo(self,users):
  ulist=','.join(users)

  fields='gender,current_location,relationship_status,'+\
         'affiliations,hometown_location'

  doc=self.sendrequest({'method':'facebook.users.getInfo',
  'session_key':self.session_key,'call_id':callid(),
  'users':ulist,'fields':fields})

  results={}
  for n,id in zip(doc.getElementsByTagName('result_elt'),users):
    # Standort ermitteln.
    locnode=n.getElementsByTagName('hometown_location')[0]
    loc=getsinglevalue(locnode,'city')+', '+getsinglevalue(locnode,'state')

    # Schule
    college=''
    gradyear='0'
    affiliations=n.getElementsByTagName('affiliations_elt')
    for aff in affiliations:
      # Typ 1 ist College.
      if getsinglevalue(aff,'type')=='1':
        college=getsinglevalue(aff,'name')
        gradyear=getsinglevalue(aff,'year')

    results[id]={'gender':getsinglevalue(n,'gender'),
                 'status':getsinglevalue(n,'relationship_status'),
                 'location':loc,'college':college,'year':gradyear}
  return results
```

Die Ergebnisse sind ein Dictionary, das die Benutzer-IDs auf eine Untermenge der Informationen abbildet. Sie können dies nutzen, um eine neue Vermittler-Datenmenge zu erstellen. Wenn Sie wollen, können Sie Ihre neue Facebook-Klasse in Ihrer Python-Session ausprobieren:

```
>>> import facebook
>>> s=facebook.fbsession()
Press enter after logging in:
>>> friends=s.getfriends()
>>> friends[1]
u'iY5TTbS-0fvs.'
>>> s.getinfo(friends[0:2])
{u'iA810MUfhfsw.': {'gender': u'Female', 'location': u'Atlanta, '},
 u'iY5TTbS-0fvs.': {'gender': u'Male', 'location': u'Boston, '}}
```

Erstellen einer Kuppel-Datenmenge

Der letzte Aufruf der Facebook-API, der für unsere Übung notwendig ist, ermittelt, ob zwei Personen Freunde sind oder nicht. Dies wird zukünftig als »Antwort« in unserer neuen Datenmenge genutzt. Der Aufruf ermöglicht es Ihnen, zwei gleich

e-bol.net

lange Listen mit IDs zu übermitteln, während Sie eine Liste mit einer Zahl für jedes Paar zurückbekommen – 1, wenn die Personen Freunde sind, ansonsten 0. Fügen Sie diese Methode Ihrer Klasse hinzu:

```
def arefriends(self,idlist1,idlist2):
  id1=','.join(idlist1)
  id2=','.join(idlist2)
  doc=self.sendrequest({'method':'facebook.friends.areFriends',
                        'session_key':self.session_key,'call_id':callid( ),
                        'id1':id1,'id2':id2})
  results=[]
  for n in doc.getElementsByTagName('result_elt'):
    results.append(n.firstChild.nodeValue)
  return results
```

Schließlich können Sie nun alles zusammenfügen, um eine Datenmenge zu erstellen, die mit der LIBSVM genutzt werden kann. Dazu wird eine Liste aller Freunde des angemeldeten Benutzers erstellt, Informationen zu diesen heruntergeladen und eine Zeile für jedes Personenpaar erstellt. Dann prüft es jedes Paar, um herauszufinden, ob sie Freunde sind. Fügen Sie `makedataset` der Klasse hinzu:

```
def makedataset(self):
  from advancedclassify import milesdistance
  # Informationen über alle meine Freunde bekommen.
  friends=self.getfriends( )
  info=self.getinfo(friends)
  ids1,ids2=[],[]
  rows=[]

  # Verschachtelte Schleife, die sich jedes Freundespaar anschaut.
  for i in range(len(friends)):
    f1=friends[i]
    data1=info[f1]

    # Bei i+1 beginnen, sodass wir nicht doppelt abfragen.
    for j in range(i+1,len(friends)):
      f2=friends[j]
      data2=info[f2]
      ids1.append(f1)
      ids2.append(f2)

      # Zahlen aus den Daten erzeugen.
      if data1['college']==data2['college']: sameschool=1
      else: sameschool=0
      male1=(data1['gender']=='Male') and 1 or 0
      male2=(data2['gender']=='Male') and 1 or 0

      row=[male1,int(data1['year']),male2,int(data2['year']),sameschool]
      rows.append(row)
  # arefriends blockweise für jedes Freundespaar aufrufen.
  arefriends=[]
  for i in range(0,len(ids1),30):
```

e-bol.net

```
        j=min(i+20,len(ids1))
        pa=self.arefriends(ids1[i:j],ids2[i:j])
        arefriends+=pa
    return arefriends,rows
```

Diese Methode wandelt das Geschlecht und den Status in Zahlen um, sodass die Datenmenge direkt von LIBSVM genutzt werden kann. Die letzte Schleife fragt den Freundesstatus jedes Personenpaars in Blöcken ab, da Facebook die Größe eines einzelnen Requests beschränkt.

Erstellen eines SVM-Modells

Um eine SVM mit Ihren Daten zu bauen, laden Sie die Klasse neu, erzeugen eine neue Session und bauen die Datenmenge auf:

```
>>> reload(facebook)
<module 'facebook' from 'facebook.pyc'>
>>> s=facebook.fbsession( )
Drücken Sie nach dem Anmelden die Return-Taste:
>>> answers,data=s.makedataset( )
```

Sie sollten nun die svm-Methoden direkt anwenden können:

```
>>> param = svm_parameter(kernel_type = RBF)
>>> prob = svm_problem(answers,data)
>>> m=svm_model(prob,param)
>>> m.predict([1,2003,1,2003,1]) # zwei Männer, gleiches Jahr, gleiches College
1.0
>>> m.predict([1,2003,1,1996,0]) # verschiedene Jahre, verschiedene Colleges
0.0
```

Natürlich werden Ihre Ergebnisse davon abweichen, aber üblicherweise wird das Modell feststellen, dass Personen, die zum gleichen College gegangen sind oder aus demselben Ort stammen, recht wahrscheinlich Freunde sind.

Übungen

1. *Bayes-Klassifizierer*. Können Sie sich Möglichkeiten vorstellen, den in Kapitel 6 gebauten Bayes-Klassifizierer für die Vermittler-Datenmenge zu nutzen? Was wären gute Beispiele für Merkmale?
2. *Optimieren einer Trennlinie*. Denken Sie, dass es möglich ist, eine Trennlinie mit den Optimierungsmethoden auszuwählen, die Sie in Kapitel 5 kennengelernt haben, statt einfach die Mittelwerte zu nutzen? Welche Zielfunktion würden Sie verwenden?
3. *Wählen der besten Kernel-Parameter*. Schreiben Sie eine Funktion, die unterschiedliche Werte von gamma durchläuft und ermittelt, welcher der beste Wert für eine gegebene Datenmenge ist.

e-bol.net

4. *Interessen-Hierarchie*. Entwerfen Sie eine einfache Interessen-Hierarchie zusammen mit einer Datenstruktur, um sie zu repräsentieren. Passen Sie die Funktion `matchcount` so an, dass sie die Hierarchie nutzt, um Teilpunkte für Übereinstimmungen zu vergeben.
5. *Andere LIBSVM-Kernel*. Schauen Sie sich die LIBSVM-Dokumentation an, um herauszufinden, welche anderen Kernel verfügbar sind. Probieren Sie einen polynomischen Kernel aus. Verbessert sich die Vorhersagequalität?
6. *Andere Facebook-Vorhersagen*. Schauen Sie sich alle Felder an, die man über die Facebook-API erreichen kann. Welche Datenmengen können für Einzelpersonen erstellt werden? Könnten Sie ein SVM-Modell nutzen, um vorherzusagen, ob jemand einen Lieblingsfilm angegeben hat – in Abhängigkeit von der Schule, zu der er ging? Was könnte man noch vorhersagen?

e-bol.net

KAPITEL 10

Unabhängige Merkmale finden

Die meisten der bisherigen Kapitel haben sich vorrangig auf *überwachte* Klassifizierer konzentriert, mit Ausnahme von Kapitel 3, in dem es um *unüberwachte* Techniken der *Clusteranalyse* ging. Dieses Kapitel wird sich Möglichkeiten vornehmen, die wichtigen zugrunde liegenden Merkmale aus Datenmengen zu ermitteln, die nicht mit bestimmten Ergebnissen verbunden sind. Wie bei der Clusteranalyse versuchen diese Methoden nicht, Vorhersagen zu treffen, sondern die Daten zu charakterisieren und Ihnen interessante Dinge über sie zu erzählen.

Sie werden aus Kapitel 3 erinnern, dass bei der Clusteranalyse jede Zeile einer Datenmenge einer Gruppe oder einem Punkt in einer Hierarchie zugewiesen wird – jedes Element passt in genau eine Gruppe, die den Durchschnitt der Mitglieder repräsentiert. Die *Merkmalsextraktion* (*Feature Extraction*) ist eine allgemeinere Form dieser Idee – sie versucht, neue Datenzeilen zu finden, die kombiniert werden können, um Zeilen der ursprünglichen Datenmenge zu rekonstruieren. Anstatt eine Zeile zu einem Cluster gehören zu lassen, wird sie aus einer Kombination aus Merkmalen erstellt.

Eines der klassischen Probleme, das die Notwendigkeit zum Finden unabhängiger Merkmale zeigt, ist als *Cocktailparty-Problem* bekannt – das Problem, Gespräche zu verfolgen, wenn viele Leute gleichzeitig reden. Ein bemerkenswertes Kennzeichen des menschlichen Gehörs ist unsere Fähigkeit, sich auf eine einzelne Stimme in einem Raum voller sprechender Personen konzentrieren zu können, obwohl unsere Ohren ein Gemisch all der verschiedenen Stimmen erreicht. Das Gehirn ist ziemlich gut darin, die verschiedenen Geräusche auseinanderzuhalten, die zusammen den »Lärmpegel« erzeugen. Durch die Verwendung von Algorithmen wie dem in diesem Kapitel beschriebenen und durch die Aufnahme der Geräusche mit mehreren Mikrofonen, die in einem Raum aufgestellt sind, ist es auch für einen Computer möglich, das Gleiche zu erreichen – eine Kakofonie an Geräuschen zu empfangen und sie auseinanderzudividieren, ohne vorher zu wissen, worum es sich dabei handelt.

e-bol.net

Eine andere interessante Anwendung der Merkmalsextraktion ist das Erkennen wiederholt vorkommender Wortmuster in einer Reihe von Dokumenten, was dabei helfen kann, Themen zu finden, die unabhängig voneinander in verschiedenen Kombinationen vorhanden sind. In diesem Kapitel werden Sie ein System aufbauen, das Nachrichten aus einer Reihe von Feeds herunterlädt und die Hauptthemen erkennt, die in dieser Sammlung von Artikeln vorkommen. Sie werden vielleicht feststellen, dass Artikel mehr als ein Thema aufweisen und Sie werden sicherlich sehen, dass Themen in mehr als einem Artikel vorkommen.

Das zweite Beispiel in diesem Kapitel dreht sich um Aktiendaten, bei denen man davon ausgeht, dass viele verschiedene zugrunde liegende Ursachen schließlich zu einem Ergebnis führen. Der gleiche Algorithmus kann auf die Daten angewendet werden, um nach diesen Ursachen und ihren unabhängigen Effekten zu suchen.

Eine Nachrichtensammlung

Zunächst einmal brauchen Sie eine Reihe von Nachrichten, mit denen Sie arbeiten können. Sie sollten aus verschiedenen Quellen stammen, damit sich die Themen, die an unterschiedlichen Orten behandelt werden, leichter erkennen lassen. Glücklicherweise bieten die meisten der großen Nachrichtensites RSS- oder Atom-Feeds an, sei es für alle Artikel oder für einzelne Kategorien. Sie haben den *Universal Feed Parser* in früheren Kapiteln schon genutzt, um RSS- und Atom-Feeds von Blogs zu parsen, und Sie können den gleichen Parser verwenden, um Nachrichten herunterzuladen. Wenn Sie ihn noch nicht haben, bekommen Sie ihn unter *http://feedparser.org*.

Quellen auswählen

Es gibt Tausende von Quellen, die als »Nachrichten« bezeichnet werden können – von den großen Sendern und Zeitungen bis zu politischen Blogs. Dazu gehören zum Beispiel (für den englischsprachigen Raum):

- Reuters
- The Associated Press
- *The New York Times*
- Google News
- Salon.com
- Fox News
- *Forbes*-Magazin
- CNN International

e-bol.net

Das ist nur eine kleine Auswahl der Möglichkeiten. Kombiniert man Quellen aus verschiedenen Bereichen des politischen Spektrums und mit unterschiedlichen Schreibstilen, hat man besseres Testmaterial für den Algorithmus, da er dazu in der Lage sein sollte, die wichtigen Merkmale herauszufinden und unwichtige Abschnitte zu ignorieren. Es ist für einen Merkmalsextraktionsalgorithmus mit den richtigen Daten auch möglich, ein Merkmal zu erkennen, das in Artikeln mit einem bestimmten politischen Einschlag vorkommt, und dieses Merkmal dem Artikel zuzuweisen – neben den Merkmalen, die das Thema des Artikels beschreiben.

Erstellen Sie eine neue Datei namens *newsfeatures.py* und fügen Sie den folgenden Code hinzu, um ein paar Bibliotheken zu importieren und eine Liste mit Ressourcen anzulegen:

```
import feedparser
import re

feedlist=['http://today.reuters.com/rss/topNews',
          'http://today.reuters.com/rss/domesticNews',
          'http://today.reuters.com/rss/worldNews',
          'http://hosted.ap.org/lineups/TOPHEADS-rss_2.0.xml',
          'http://hosted.ap.org/lineups/USHEADS-rss_2.0.xml',
          'http://hosted.ap.org/lineups/WORLDHEADS-rss_2.0.xml',
          'http://hosted.ap.org/lineups/POLITICSHEADS-rss_2.0.xml',
          'http://www.nytimes.com/services/xml/rss/nyt/HomePage.xml',
          'http://www.nytimes.com/services/xml/rss/nyt/International.xml',
          'http://news.google.com/?output=rss',
          'http://feeds.salon.com/salon/news',
          'http://www.foxnews.com/xmlfeed/rss/0,4313,0,00.rss',
          'http://www.foxnews.com/xmlfeed/rss/0,4313,80,00.rss',
          'http://www.foxnews.com/xmlfeed/rss/0,4313,81,00.rss',
          'http://rss.cnn.com/rss/edition.rss',
          'http://rss.cnn.com/rss/edition_world.rss',
          'http://rss.cnn.com/rss/edition_us.rss']
```

Die Feedliste enthält verschiedenste Quellen, vor allem Topnews, Nachrichten aus aller Welt und aus den USA. Sie können die Feedliste so anpassen, dass Ihre eigenen Wunschquellen darin auftauchen, aber Sie sollten schon ein paar überlappende Bereiche haben. Wenn keiner der Artikel etwas gemeinsam hat, wird es für den Algorithmus sehr schwierig werden, wichtige Merkmale zu extrahieren, und Sie werden bei Merkmalen enden, die nicht wirklich etwas bedeuten.

Quellen herunterladen

Der Merkmalsextraktionsalgorithmus erwartet wie der Algorithmus zur Clusteranalyse eine große Matrix mit Zahlen, bei der jede Zeile für ein Element und jede Spalte für eine Eigenschaft steht. In diesem Fall werden die verschiedenen Artikel die Zeilen und die Wörter die Spalten sein. Die Zahlen werden repräsentieren, wie oft ein Wort in einem gegebenen Artikel vorkommt. Daher wird Ihnen die folgende Matrix

e-bol.net

zeigen, dass Artikel A das Wort »hurricane« drei Mal enthält, Artikel B das Wort »democrats« zwei Mal und so weiter.

```
articles = ['A','B','C',...
words = ['hurricane','democrats','world',...
matrix = [[3,0,1,...]
          [1,2,0,...]
          [0,0,2,...]
                ...]
```

Um aus einem Feed eine Matrix dieser Form zu erhalten, brauchen Sie ein paar Methoden, die denen aus früheren Kapiteln ähneln. Die erste Methode entfernt alle Bilder und Markups aus den Artikeln. Fügen Sie `stripHTML` der Datei *newsfeatures.py* hinzu:

```
def stripHTML(h):
  p=''
  s=0
  for c in h:
    if c=='<': s=1
    elif c=='>':
      s=0
      p+=' '
    elif s==0: p+=c
  return p
```

Sie werden auch eine Möglichkeit brauchen, die Wörter im Text zu separieren, wie Sie es bereits weiter oben in diesem Buch getan haben. Wenn Sie einen ausgereifteren Weg entwickelt haben, Wörter zu trennen, und nicht nur den einfachen alphanumerischen regulären Ausdruck nutzen wollen, können Sie die Funktion hier wiederverwenden. Ansonsten fügen Sie diese Funktion *newsfeatures.py* hinzu:

```
def separatewords(text):
  splitter=re.compile('\\W*')
  return [s.lower( ) for s in splitter.split(text) if len(s)>3]
```

Die nächste Funktion läuft über alle Feeds, parst sie mit dem `feedparser`, schmeißt das HTML-Coding raus und extrahiert die einzelnen Wörter mit den weiter oben definierten Funktionen. Sie merkt sich, wie oft jedes Wort insgesamt verwendet wurde und wie oft es in jedem einzelnen Artikel vorkommt.

Fügen Sie diese Funktion ebefalls dieser *newsfeatures.py* hinzu:

```
def getarticlewords( ):
  allwords={}
  articlewords=[]
  articletitles=[]
  ec=0
  # Über jeden Feed laufen.
  for feed in feedlist:
    f=feedparser.parse(feed)

    # Über jeden Artikel laufen.
```

e-bol.net

```
    for e in f.entries:
      # Identische Artikel ignorieren.
      if e.title in articletitles: continue

      # Die Wörter extrahieren.
      txt=e.title.encode('utf8')+stripHTML(e.description.encode('utf8'))
      words=separatewords(txt)
      articlewords.append({})
      articletitles.append(e.title)

      # Die Zähler für diese Wörter in allwords und articlewords erhöhen.
      for word in words:
        allwords.setdefault(word,0)
        allwords[word]+=1
        articlewords[ec].setdefault(word,0)
        articlewords[ec][word]+=1
      ec+=1
  return allwords,articlewords,articletitles
```

Die Funktion hat drei Variablen:

- `allwords` merkt sich, wie oft die Wörter in allen Artikeln zusammen genutzt wurden. Das wird verwendet, um zu bestimmen, welche Wörter als Teil von Merkmalen in Betracht gezogen werden sollten.
- `articlewords` enthält die Zähler für die Wörter in jedem Artikel.
- `articletitles` ist eine Liste mit Titeln der Artikel.

Umwandeln in eine Matrix

Jetzt haben Sie Dictionaries mit Zählern für die Wörter in allen Artikeln und in jedem einzelnen Artikel, die nun in die vorher beschriebene Matrix konvertiert werden müssen. Im ersten Schritt wird nun eine Liste mit Wörtern erstellt, die als Spalten der Matrix verwendet werden. Um die Größe der Matrix zu reduzieren, können Sie Wörter herausschmeißen, die nur in ein paar Artikeln vorkommen (und damit sehr wahrscheinlich für das Finden von Merkmalen nicht sehr hilfreich sein werden) oder in zu vielen Artikeln erscheinen.

Versuchen Sie zu Beginn, nur Wörter aufzunehmen, die in mehr als drei Artikeln, aber in weniger als 60% der Artikel vorkommen. Sie können dann eine verschachtelte *List Comprehension* nutzen, um die Matrix zu erzeugen, die auch nicht mehr als eine Liste mit Listen ist. Jede der verschachtelten Listen wird erzeugt, indem man über `wordvec` läuft und im Dictionary nach dem Wort sucht. Fehlt das Wort, wird eine 0 hinzugefügt, ansonsten wird der Wortzähler für diesen Artikel und dieses Wort hinzuaddiert.

Fügen Sie die Funktion `makematrix` *newsfeatures.py* hinzu:

```
def makematrix(allw,articlew):
  wordvec=[]
```

e-bol.net

```
    # Nur Wörter, die häufig, aber nicht zu häufig vorkommen.
    for w,c in allw.items( ):
      if c>3 and c<len(articlew)*0.6:
        wordvec.append(w)

    # Erstellen der Wort-Matrix.
    l1=[[(word in f and f[word] or 0) for word in wordvec] for f in articlew]
    return l1,wordvec
```

Starten Sie eine Python-Session und importieren Sie `newsfeatures`. Sie können dann versuchen, die Feeds zu parsen und die Matrix zu erstellen.

```
$ python
>>> import newsfeatures
>>> allw,artw,artt= newsfeatures.getarticlewords( )
>>> wordmatrix,wordvec= newsfeatures.makematrix(allw,artw)
>>> wordvec[0:10]
['increase', 'under', 'regan', 'rise', 'announced', 'force',
 'street', 'new', 'men', 'reported']
>>> artt[1]
u'Fatah, Hamas men abducted freed: sources'
>>> wordmatrix[1][0:10]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
```

In diesem Beispiel werden die ersten zehn Wörter des Wortvektors angezeigt. Der Titel des zweiten Artikels wird ausgegeben, gefolgt von den ersten zehn Werten seiner Zeile in der Wort-Matrix. Sie können erkennen, dass der Artikel das Wort »men« einmal enthält, jedoch keines der anderen Wörter aus den ersten zehn Werten.

Vorangegangene Ansätze

In früheren Kapiteln haben Sie unterschiedliche Wege genutzt, um mit Worthäufigkeiten bei Textdaten umzugehen. Es ist nützlich, diese zuerst auszuprobieren und zu sehen, was für Ergebnisse Sie erhalten, um sie dann mit den Ergebnissen aus der Merkmalsextraktion zu vergleichen. Wenn Sie den Code aus diesen Kapiteln noch haben, können Sie die Module importieren und sie auf Ihre Feeds anwenden. Wenn nicht, ist es auch nicht schlimm – dieser Abschnitt wird erklären, wie die Methoden mit den Beispieldaten arbeiten.

Bayes-Klassifizierer

Die *Bayes-Klassifikation* ist, wie Sie gesehen haben, eine überwachte Lernmethode. Wollten Sie versuchen, den in Kapitel 6 gebauten Klassifizierer einzusetzen, müssten Sie zunächst einmal viele Nachrichtenbeispiele nutzen, um den Klassifizierer zu trainieren. Der Klassifizierer wäre dann dazu in der Lage, später folgende Artikel in Ihre vordefinierten Kategorien einzuordnen. Neben dem offensichtlichen Nachteil, das Training durchführen zu müssen, gibt es noch das Problem, dass der Entwickler entscheiden muss, welche Kategorien es gibt. Alle bisher vorgestellten Klassifizierer,

e-bol.net

wie zum Beispiel die Entscheidungsbäume und die Support-Vektor-Maschinen, werden die gleichen Einschränkungen haben, wenn man sie auf solch eine Datenmenge loslässt.

Wenn Sie den Bayes-Klassifizierer für diese Datenmenge ausprobieren wollen, müssen Sie das in Kapitel 6 entwickelte Modul in Ihrem Arbeitsverzeichnis unterbringen. Sie können das Dictionary `articlewords` als Merkmalsset jedes Artikels verwenden.

Probieren Sie dies in Ihrer Python-Session aus:

```
>>> def wordmatrixfeatures(x):
...     return [wordvec[w] for w in range(len(x)) if x[w]>0]
...
>>> wordmatrixfeatures(wordmatrix[0])
['forces', 'said', 'security', 'attacks', 'iraq', 'its', 'pentagon',...]
>>> import docclass
>>> classifier=docclass.naivebayes(wordmatrixfeatures)
>>> classifier.setdb('newstest.db')
>>> artt[0]
u'Attacks in Iraq at record high: Pentagon'
>>> # Als Irak-Artikel trainieren.
>>> classifier.train(wordmatrix[0],'iraq')
>>> artt[1]
u'Bush signs U.S.-India nuclear deal'
>>> # Als Indien-Artikel trainieren.
>>> classifier.train(wordmatrix[1],'india')
>>> artt[2]
u'Fatah, Hamas men abducted freed: sources'
>>> # Wie wird dieser Artikel klassifiziert?
>>> classifier.classify(wordmatrix[1])
u'iraq'
```

Bei den verwendeten Beispielen gibt es viele mögliche Themen und in jedem von ihnen nur ein paar Artikel. Der Bayes-Klassifizierer wird schließlich alle diese Themen lernen, aber da man viele Beispiele pro Thema trainieren muss, passt dieser Klassifizierer besser zu weniger Kategorien mit mehr Beispielen in jeder Kategorie.

Clusteranalyse

Den anderen, unüberwachten Ansatz, den Sie kennengelernt haben, ist die Clusteranalyse, die wir in Kapitel 3 behandelten.

Die Daten in Kapitel 3 wurden in einer Matrix angeordnet, die mit der weiter oben erstellten identisch ist. Wenn Sie den Code aus diesem Kapitel noch haben, können Sie ihn in Ihre Python-Session importieren und den Clusteralgorithmus für die Matrix ausführen:

```
>>> import clusters
>>> clust=clusters.hcluster(wordmatrix)
>>> clusters.drawdendrogram(clust,artt,jpeg='news.jpg')
```

e-bol.net

Abbildung 10-1 zeigt ein mögliches Ergebnis der Clusteranalyse, das in einer Datei namens *news.jpg* gespeichert wurde.

Abbildung 10-1: Dendrogramm mit den aufgeteilten Nachrichten

e-bol.net

Wie erwartet, werden ähnliche Nachrichtenartikel zusammen gruppiert. Das funktioniert sogar noch besser als das Blog-Beispiel aus Kapitel 3, da viele Magazine dazu tendieren, die gleichen Geschichten in einer ähnlichen Sprache zu erzählen. Aber ein paar Beispiele in Abbildung 10-1 zeigen, dass Sie nicht immer ein genaues Bild des Ganzen erhalten, wenn Sie Nachrichtenartikel in einzelne Kategorien unterteilen. So ist zum Beispiel »The Nose Knows Better« – ein Artikel aus dem Gesundheitsbereich – zusammen mit einem Artikel über den Suffolk Strangler (ein englischer Serienmörder) gruppiert. Manchmal lassen sich Nachrichtenartikel genau wie Personen eben nicht in eine einzelne Schublade stecken und müssen als einzigartig angesehen werden.

Wenn Sie möchten, können Sie die Matrix rotieren, um zu sehen, wie die Wörter in den Nachrichten zusammengehören. In diesem Beispiel liegen Wörter wie »station«, »solar« und »astronauts« nahe beieinander.

Nicht-negative Matrix-Faktorisierung

Die Technik zum Extrahieren der wichtigen Merkmale aus den Daten wird als *Nicht-negative Matrix-Faktorisierung* (*Non-Negative Matrix Factorization*, NMF) bezeichnet. Das ist eine der kompliziertesten Techniken, die in diesem Buch behandelt werden, und erfordert ein wenig mehr Erläuterung und eine kurze Einführung in die lineare Algebra. Alles, was Sie wissen müssen, wird nun in diesem Abschnitt erklärt.

Eine kurze Einführung in die Matrizen-Mathematik

Um zu verstehen, was die NMF tut, müssen Sie erst ein wenig über Matrix-Multiplikationen wissen. Wenn Sie schon mit linearer Algebra vertraut sind, können Sie unbesorgt zum nächsten Abschnitt springen.

Ein Beispiel für eine Matrix-Multiplikation ist in Abbildung 10-2 zu sehen.

$$\underset{\mathbf{A}}{\begin{bmatrix} 1 & 4 \\ 0 & 3 \end{bmatrix}} \mathbf{X} \underset{\mathbf{B}}{\begin{bmatrix} 0 & 30 \\ 2 & 14 \end{bmatrix}} = \begin{bmatrix} 1*0 + 4*2 & 1*3 + 4*1 & 1*0 + 4*4 \\ 0*0 + 3*2 & 0*3 + 3*1 & 0*0 + 3*4 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 16 \\ 6 & 3 & 12 \end{bmatrix}$$

Abbildung 10-2: Beispiel für eine Matrix-Multiplikation

Diese Abbildung zeigt, wie zwei Matrizen miteinander multipliziert werden. Dabei muss die Anzahl der Spalten der ersten Matrix (in diesem Beispiel Matrix A) gleich der Anzahl an Zeilen der zweiten Matrix (Matrix B) sein. In diesem Fall hat Matrix A zwei Spalten und Matrix B zwei Zeilen. Die Ergebnis-Matrix (Matrix C) wird die

e-bol.net

gleiche Anzahl an Zeilen wie Matrix A und die gleiche Anzahl an Spalten wie Matrix B haben.

Der Wert jeder Zelle in Matrix C wird ermittelt, indem man die Werte aus der gleichen Zeile von Matrix A mit den Werten aus der gleichen Spalte von Matrix B multipliziert und alle Produkte addiert. Schaut man sich den Wert in der oberen linken Ecke von Matrix C an, sieht man, dass die Werte aus der ersten Zeile von Matrix A mit den entsprechenden Werten aus der ersten Spalte von Matrix B multipliziert werden. Dann werden sie addiert, um das Endergebnis zu erhalten. Die anderen Zellen in Matrix C werden genau so berechnet.

Eine andere häufig vorkommende Matrix-Operation ist das *Transponieren*. Das bedeutet, dass die Spalten zu Zeilen und die Zeilen zu Spalten werden. Diese Operation wird normalerweise mit einem »T« angegeben, wie in Abbildung 10-3 zu sehen.

$$\begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}^T = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

Abbildung 10-3: Transponieren einer Matrix

Sie werden das Transponieren und Multiplizieren bei der Implementierung des NMF-Algorithmus nutzen.

Was hat das mit der Artikel-Matrix zu tun?

Bisher haben Sie nur eine Matrix mit Artikeln und Wortfrequenzen. Das Ziel ist, diese Matrix zu *faktorisieren*, also zwei kleinere Matrizen zu finden, die miteinander multipliziert werden können, um diese eine zu rekonstruieren. Die zwei kleineren Matrizen sind:

Die Merkmals-Matrix
: Diese Matrix hat eine Zeile für jedes Merkmal und eine Spalte für jedes Wort. Die Werte geben an, wie wichtig ein Wort für ein Merkmal ist. Jedes Merkmal sollte ein Thema repräsentieren, das aus einer Reihe von Artikeln extrahiert wurde, daher können Sie erwarten, dass ein Artikel über eine neue Fernsehsendung eine hohe Gewichtung des Worts »television« haben wird.

Die Gewichtungs-Matrix
: Diese Matrix bildet die Merkmale auf die Artikel-Matrix ab. Jede Zeile ist ein Artikel und jede Spalte ein Merkmal. Die Werte geben an, wie sehr jedes Merkmal auf jeden Artikel passt.

e-bol.net

Die Merkmals-Matrix hat wie die Artikel-Matrix eine Spalte für jedes Wort. Jede Zeile der Matrix ist ein anderes Merkmal, daher ist ein Merkmal eine Liste mit Wortgewichtungen. Abbildung 10-4 zeigt einen Beispielausschnitt einer Merkmals-Matrix.

$$\begin{array}{r} \\ \text{Merkmal 1} \\ \text{Merkmal 2} \\ \text{Merkmal 3} \end{array} \begin{array}{c} \begin{array}{cccc} \text{hurricane} & \text{democrats} & \text{florida} & \text{elections} \end{array} \\ \begin{pmatrix} 2 & 0 & 3 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Abbildung 10-4: Ausschnitt aus einer Merkmals-Matrix

Da jede Zeile ein Merkmal ist, das aus einer Kombination aus Wörtern besteht, sollte es klar sein, dass das Rekonstruieren der Artikel-Matrix eine Frage der Kombination von Zeilen aus der Merkmals-Matrix in unterschiedlicher Gewichtung ist. Die Gewichtungs-Matrix, von der ein Beispiel in Abbildung 10-5 zu sehen ist, bildet die Merkmale auf die Artikel ab. Sie hat eine Spalte für jedes Merkmal und eine Zeile für jeden Artikel.

$$\begin{array}{r} \\ \text{hurricane in Florida} \\ \text{Democrats sweep elections} \\ \text{Democrats dispute Florida ballots} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Merkmal 1} & \text{Merkmal 2} & \text{Merkmal 3} \end{array} \\ \begin{pmatrix} 10 & 0 & 0 \\ 0 & 8 & 1 \\ 0 & 5 & 6 \end{pmatrix} \end{array}$$

Abbildung 10-5: Ausschnitt aus der Gewichtungs-Matrix

Abbildung 10-6 zeigt, wie die Artikel-Matrix rekonstruiert wird, indem die Merkmals-Matrix mit der Gewichtungs-Matrix multipliziert wird.

$$\begin{array}{r} \\ \text{M1} \\ \text{M2} \\ \text{M3} \end{array} \begin{array}{c} \begin{array}{cccc} \text{hurricane} & \text{democrats} & \text{florida} & \text{elections} \end{array} \\ \begin{pmatrix} 2 & 0 & 3 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \times \begin{array}{r} \\ \text{hurricane…} \\ \text{…sweep…} \\ \text{Florida ballots} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Merkmal 1} & \text{Merkmal 2} & \text{Merkmal 3} \end{array} \\ \begin{pmatrix} 10 & 0 & 0 \\ 0 & 8 & 1 \\ 0 & 5 & 6 \end{pmatrix} \end{array} = \begin{array}{r} \\ \text{hurricane…} \\ \text{…sweep…} \\ \text{Florida ballots} \end{array} \begin{array}{c} \begin{array}{cccc} \text{hurricane} & \text{democrats} & \text{florida} & \text{elections} \end{array} \\ \begin{pmatrix} \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad \end{pmatrix} \end{array}$$

Abbildung 10-6: Multiplizieren der Merkmals-Matrix mit der Gewichtungs-Matrix

Wenn die Anzahl der Merkmale genauso groß ist wie die Anzahl der Artikel, ist die beste Reaktion, ein Merkmal zu haben, das perfekt zu einem Artikel passt. Aber der Zweck der Matrix-Faktorisierung ist hier, eine große Menge an Beobachtungen (in diesem Fall Artikel) auf eine kleinere Menge zu reduzieren, die die gemeinsamen Merkmale umfasst. Idealerweise kann diese kleinere Menge an Merkmalen mit unterschiedlichen Gewichtungen so kombiniert werden, dass die ursprüngliche Datenmenge perfekt reproduziert wird, aber in der Praxis ist dies sehr unwahr-

e-bol.net

scheinlich, daher versucht der Algorithmus, die Datenmenge zumindest so gut wie möglich wiederherzustellen.

Diese Form der Matrix-Faktorisierung wird *nicht-negativ* genannt, weil sie Merkmale und Gewichtungen ohne negative Werte zurückgibt. Das bedeutet also, dass alle Merkmale positive Werte haben müssen, was für unser Beispiel gültig ist, da ein Artikel keine negative Wortanzahlen haben kann. Es bedeutet auch, dass Merkmale keine anderen Merkmale reduzieren können – die NMF wird keine Ergebnisse liefern, die explizit bestimmte Wörter ausschließen. Auch wenn diese Einschränkung den Algorithmus davon abhalten kann, die bestmögliche Faktorisierung zurückzuliefern, lassen sich die Ergebnisse häufig besser interpretieren.

Verwenden von NumPy

Python hat keine mitgelieferten Funktionen für Matrix-Operationen. Man kann sie sich zwar selbst schreiben, aber es ist besser, ein Paket namens *NumPy* zu installieren. Es bietet nicht nur ein Matrix-Objekt und alle notwendigen Matrix-Operationen an, sondern kann sich in Bezug auf die Performance auch noch mit kommerzieller mathematischer Software messen. Sie können NumPy von *http://numpy.scipy.org* herunterladen.

Mehr Informationen zur Installation von NumPy finden Sie in Anhang A.

NumPy stellt ein Matrix-Objekt bereit, das mit verschachtelten Listen instantiiert werden kann und sehr demjenigen ähnelt, das Sie für die Artikel erstellt haben. Um dies einmal auszuprobieren, können Sie NumPy in Ihrer Python-Session importieren und eine Matrix erstellen:

```
>>> from numpy import *
>>> l1=[[1,2,3],[4,5,6]]
>>> l1
[[1, 2, 3], [4, 5, 6]]
>>> m1=matrix(l1)
>>> m1
matrix([[1, 2, 3],
        [4, 5, 6]])
```

Die Objekte vom Typ `matrix` stellen mathematische Operationen wie die Multiplikation und Addition mit Standardoperatoren bereit. Das Transponieren der Matrix führt man mit der Funktion `transpose` durch.

```
>>> m2=matrix([[1,2],[3,4],[5,6]])
>>> m2
matrix([[1, 2],
        [3, 4],
        [5, 6]])
>>> m1*m2
matrix([[22, 28],
        [49, 64]])
```

e-bol.net

Die Funktion shape liefert die Anzahl an Zeilen und Spalten der Matrix zurück, was nützlich sein kann, wenn man mit Schleifen über alle Elemente einer Matrix laufen will:

```
>>> shape(m1)
(2, 3)
>>> shape(m2)
(3, 2)
```

Schließlich bietet NumPy auch noch ein schnelles array-Objekt an, das wie eine Matrix mehrdimensional sein kann. Matrizen lassen sich einfach in Arrays umwandeln und umgekehrt. Ein Array verhält sich beim Multiplizieren anders als eine Matrix – Arrays können nur multipliziert werden, wenn sie genau die gleichen Ausmaße haben, und jeder Wert wird einfach mit dem entsprechenden Wert im anderen Array multipliziert. Daher:

```
>>> a1=m1.A
>>> a1
array([[1, 2, 3],
       [4, 5, 6]])
>>> a2=array([[1,2,3],[1,2,3]])
>>> a1*a2
array([[ 1,  4,  9],
       [ 4, 10, 18]])
```

Die hohe Geschwindigkeit von NumPy ist für den NMF-Algorithmus, den Sie jetzt kennenlernen werden, notwendig, da viele Matrix-Operationen durchgeführt werden.

Der Algorithmus

Der Algorithmus, den Sie zum Faktorisieren der Matrix nutzen werden, wurde erstmals in den späten 1990ern veröffentlicht, womit er der jüngste Algorithmus ist, der in diesem Buch behandelt wird. Es hat sich gezeigt, dass er bei bestimmten Problemen sehr gut funktioniert, zum Beispiel beim automatischen Erkennen verschiedener Gesichtsmerkmale in einer Reihe von Fotos.

Der Algorithmus versucht, die Artikel-Matrix so gut wie möglich zu rekonstruieren, indem er die beste Merkmals- und Gewichtungs-Matrix berechnet. In diesem Fall ist es nützlich, messen zu können, wie stark sich das Ergebnis dem Original nähert. Die Funktion difcost durchläuft jeden Wert in zwei gleich großen Matrizen und addiert die Quadrate der Differenzen der Einzelwerte auf.

Erstellen Sie eine neue Datei namens *nmf.py* und fügen Sie die Funktion difcost hinzu:

```
from numpy import *

def difcost(a,b):
  dif=0
```

e-bol.net

```
  # Jede Zeile und Spalte in der Matrix durchlaufen.
  for i in range(shape(a)[0]):
    for j in range(shape(a)[1]):
      # Differenzen addieren.
      dif+=pow(a[i,j]-b[i,j],2)
  return dif
```

Nun brauchen Sie noch eine Möglichkeit, die Matrizen allmählich anzupassen, um die Zielfunktion zu reduzieren. Wenn Sie Kapitel 5 gelesen haben, werden Sie feststellen, dass Sie eine Zielfunktion haben und daher auf jeden Fall eine Abkühlungs- oder Schwarm-Optimierung einsetzen können, um eine gute Lösung zu finden. Allerdings gibt es einen deutlich effizienteren Weg, nach guten Lösungen zu suchen, nämlich die Verwendung der *Multiplicative Update Rules*.

Die Herleitung dieser Regeln würde den Rahmen dieses Kapitels sprengen, aber wenn Sie daran interessiert sind, mehr über sie zu erfahren, können Sie den Originalartikel unter *http://hebb.mit.edu/people/seung/papers/nmfconverge.pdf* einsehen.

Die Regeln erzeugen vier neue Update-Matrizen. In der folgenden Beschreibung wird die ursprüngliche Artikel-Matrix als *Daten-Matrix* bezeichnet:

hn
: Die transponierte Gewichtungs-Matrix, multipliziert mit der Daten-Matrix.

hd
: Die transponierte Gewichtungs-Matrix, multipliziert mit der Gewichtungs-Matrix und der Merkmals-Matrix.

wn
: Die Daten-Matrix multipliziert mit der transponierten Merkmals-Matrix.

wd
: Die Gewichtungs-Matrix, multipliziert mit der Merkmals-Matrix und der transponierten Merkmals-Matrix.

Um die Merkmals- und Gewichtungs-Matrizen anzupassen, werden all diese Matrizen in Arrays umgewandelt. Jeder Wert in der Merkmals-Matrix wird mit dem entsprechenden Wert in *hn* multipliziert und durch den entsprechenden Wert in *hd* dividiert. Genauso wird jeder Wert in der Gewichtungs-Matrix mit dem entsprechenden Wert in *wn* multipliziert und durch den Wert in *wd* dividiert.

Die Funktion `factorize` führt diese Berechnungen durch. Fügen Sie `factorize` der Datei *nmf.py* hinzu:

```
def factorize(v,pc=10,iter=50):
  ic=shape(v)[0]
  fc=shape(v)[1]

  # Inititialisiert die Gewichtungs- und Merkmals-Matrizen mit Zufallswerten.
  w=matrix([[random.random( ) for j in range(pc)] for i in range(ic)])
  h=matrix([[random.random( ) for i in range(fc)] for i in range(pc)])
```

e-bol.net

```
  # Operationen maximal iter Mal durchführen.
  for i in range(iter):
    wh=w*h

    # Berechnen des aktuellen Fehlers.
    cost=difcost(v,wh)

    if i%10==0: print cost

    # Abbrechen, wenn die Matrix vollständig faktorisiert wurde.
    if cost==0: break

    # Merkmals-Matrix aktualisieren.
    hn=(transpose(w)*v)
    hd=(transpose(w)*w*h)

    h=matrix(array(h)*array(hn)/array(hd))

    # Gewichtungs-Matrix aktualisieren.
    wn=(v*transpose(h))
    wd=(w*h*transpose(h))

    w=matrix(array(w)*array(wn)/array(wd))

  return w,h
```

Die Funktion zum Faktorisieren der Matrix erwartet von Ihnen eine Angabe der Anzahl der Merkmale, die Sie finden möchten. In manchen Fällen werden Sie wissen, wie viele Merkmale Sie haben wollen (zwei Stimmen bei einer Aufnahme oder fünf Hauptnachrichtenthemen pro Tag), aber in anderen Fällen werden Sie nicht wissen, wie viele anzugeben sind. Es gibt im Allgemeinen keine Möglichkeit, automatisch die korrekte Anzahl an Komponenten zu bestimmen, aber durch Experimentieren kann man eventuell den passenden Bereich herausfinden.

Sie können versuchen, diese Funktion auf die Matrix `m1*m2` aus Ihrer Session anzuwenden, um herauszufinden, ob der Algorithmus eine Lösung findet, die der Ursprungs-Matrix ähnlich ist:

```
>>> import nmf
>>> w,h= nmf.factorize(m1*m2,pc=3,iter=100)
7632.94395925
0.0364091326734
...
1.12810164789e-017
6.8747907867e-020
>>> w*h
matrix([[ 22.,  28.],
        [ 49.,  64.]])
>>> m1*m2
matrix([[22, 28],
        [49, 64]])
```

e-bol.net

Der Algorithmus schafft es, eine Gewichtungs- und eine Merkmals-Matrix zu finden, die, multipliziert miteinander, das Original genau wiedergeben. Sie können dies auch mit der Artikel-Matrix ausprobieren, um zu sehen, wie gut die wichtigen Merkmale extrahiert werden können (das kann einige Zeit dauern):

```
>>> v=matrix(wordmatrix)
>>> weights,feat=nmf.factorize(v,pc=20,iter=50)
1712024.47944
2478.13274637
2265.75996871
2229.07352131
2211.42204622
```

Die Variable `feat` enthält nun die Merkmale der Nachrichtenartikel, während `weights` die Werte enthält, die angeben, wie sehr jedes Merkmal auf einen Artikel passt. Allerdings wird es Ihnen nicht sehr viel bringen, die Matrix anzustarren, daher brauchen Sie noch eine Möglichkeit, die Ergebnisse ordentlich auszugeben und interpretieren zu können.

Anzeigen der Ergebnisse

Es ist nicht so einfach zu entscheiden, wie die Ergebnisse ausgegeben werden sollen. Jedes Merkmal in der Merkmals-Matrix hat eine Gewichtung, die angibt, wie stark jedes Wort zu diesem Merkmal passt. Daher können Sie versuchen, die »stärksten« fünf oder zehn Wörter für jedes Merkmal auszugeben, um herauszufinden, worauf sich dieses Merkmal bezieht. Die entsprechende Spalte in der Gewichtungs-Matrix teilt Ihnen mit, wie stark dieses jeweilige Merkmal auf jeden der Artikel zutrifft, daher ist es auch interessant, die besten drei Artikel anzuzeigen, um zu sehen, wie sehr dieses Merkmal auf alle von ihnen zutrifft.

Fügen Sie eine neue Funktion namens `showfeatures` der Datei *newsfeatures.py* hinzu:

```
from numpy import *
def showfeatures(w,h,titles,wordvec,out='features.txt'):
  outfile=file(out,'w')
  pc,wc=shape(h)
  toppatterns=[[] for i in range(len(titles))]
  patternnames=[]

  # Über alle Merkmale laufen.
  for i in range(pc):
    slist=[]
    # Eine Liste mit Wörtern und ihren Gewichtungen erstellen.
    for j in range(wc):
      slist.append((h[i,j],wordvec[j]))
    # Die Wortliste absteigend sortieren.
    slist.sort()
    slist.reverse()
```

e-bol.net

```
    # Ausgabe der ersten sechs Elemente.
    n=[s[1] for s in slist[0:6]]
    outfile.write(str(n)+'\n')
    patternnames.append(n)

    # Erstellen einer Artikelliste für dieses Merkmal.
    flist=[]
    for j in range(len(titles)):
      # Artikel mit Gewichtungen hinzufügen.
      flist.append((w[j,i],titles[j]))
      toppatterns[j].append((w[j,i],i,titles[j]))

    # Liste absteigend sortieren.
    flist.sort()
    flist.reverse()

    # Die obersten drei Artikel anzeigen.
    for f in flist[0:3]:
      outfile.write(str(f)+'\n')
    outfile.write('\n')

  outfile.close()
  # Musternamen für spätere Verwendung zurückgeben.
  return toppatterns,patternnames
```

Diese Funktion läuft über alle Merkmale und erstellt eine Liste mit allen Wortgewichtungen und Wörtern aus dem Wortvektor. Dann sortiert sie die Liste, sodass das am stärksten gewichtete Wort für das Merkmal am Anfang der Liste steht, und druckt die ersten sechs dieser Wörter aus. Dadurch sollten Sie das Thema grob erkennen können, das durch dieses Merkmal repräsentiert wird. Die Funktion liefert die Topmuster und die Musternamen zurück, sodass sie nur einmal berechnet werden müssen und dann in der weiter unten definierten Funktion `showarticles` wiederverwendet werden können.

Nach dem Anzeigen des Merkmals durchläuft die Funktion die Titel der Artikel und sortiert sie aufgrund der Werte in der Gewichtungs-Matrix für diesen Artikel und dieses Merkmal. Dann druckt sie die drei Artikel aus, die am stärksten mit dem Merkmal verbunden sind, wobei auch die Werte der Gewichtungs-Matrix angegeben werden. Sie werden sehen, dass ein Merkmal manchmal für viele miteinander in Verbindung stehende Artikel wichtig ist, während es ein anderes Mal nur bei einem Artikel passt.

Sie können nun diese Funktion aufrufen, um die Merkmale anschauen zu können:

```
>>> reload(newsfeatures)
<module 'newsfeatures' from 'newsfeatures.py'>
>>> topp,pn= newsfeatures.showfeatures(weights,feat,artt,wordvec)
```

Da die Ergebnisse recht umfangreich sind, wird der Code sie in einer Textdatei ablegen. Die Funktion wurde angewiesen, 20 verschiedene Merkmale zu erzeugen. Es gibt natürlich mehr als 20 Themen unter den Hunderten von Artikeln, aber die

e-bol.net

wichtigsten wurden hoffentlich extrahiert. Das Ergebnis könnte zum Beispiel so beginnen:

```
[u'palestinian', u'elections', u'abbas', u'fatah', u'monday', u'new']
(14.189453058041485, u'US Backs Early Palestinian Elections - ABC News')
(12.748863898714507, u'Abbas Presses for New Palestinian Elections Despite
Violence')
(11.286669969240645, u'Abbas Determined to Go Ahead With Vote')
```

Dieses Merkmal hat eindeutig eine Reihe von Wörtern, die mit Wahlen in Palästina zu tun haben, wobei ein paar sehr passende Artikel dazugehören. Da die Ergebnisse durch Teile des Nachrichtentexts, aber auch durch den Titel bestimmt werden, können Sie erkennen, dass sowohl der erste als auch der dritte Artikel mit diesem Merkmal verbunden sind, obwohl sie keine gemeinsamen Wörter im Titel haben. Da zudem die wichtigsten Begriffe Wörtern entstammen, die in vielen Artikeln verwendet werden, tauchen die Wörter »palestinian« und »elections« an erster Stelle auf.

Bei manchen Merkmalen gibt es eine solche klare Sammlung von zugehörigen Artikeln nicht, aber sie sind immer noch für interessante Ergebnisse gut – zum Beispiel hier:

```
[u'cancer', u'fat', u'low', u'breast', u'news', u'diet']
(29.808285029040864, u'Low-Fat Diet May Help Breast Cancer')
(2.3737882572527238, u'Big Apple no longer Fat City')
(2.3430261571622881, u'The Nose Knows Better')
```

Dieses Merkmal ist ganz offensichtlich eng mit einem Artikel über Brustkrebs verknüpft. Aber bei den schwächeren Verbindungen gibt es trotzdem Artikel zum Thema Gesundheit, die mit dem ersten Artikel ein paar Wörter gemeinsam haben.

Anzeige nach Artikel

Eine weitere Möglichkeit, die Daten anzuzeigen, ist die Ausgabe jedes Artikels und der drei Hauptmerkmale, die mit ihm verbunden sind. Damit können Sie feststellen, ob ein Artikel mehrere gleichwertige Themen oder ein einzelnes Hauptthema behandelt.

Fügen Sie eine neue Funktion `showarticles` der Datei *newsfeatures.py* hinzu:

```
def showarticles(titles,toppatterns,patternnames,out='articles.txt'):
  outfile=file(out,'w')

  # Über alle Artikel laufen.
  for j in range(len(titles)):
    outfile.write(titles[j].encode('utf8')+'\n')

    # Hauptmerkmale für diesen Artikel holen und sie
    # absteigend sortieren.
    toppatterns[j].sort( )
```

e-bol.net

```
    toppatterns[j].reverse()

    # Die ersten drei Muster ausgeben.
    for i in range(3):
      outfile.write(str(toppatterns[j][i][0])+' '+
                    str(patternnames[toppatterns[j][i][1]])+'\n')
    outfile.write('\n')

  outfile.close()
```

Da die Hauptmerkmale für jeden Artikel schon von `showfeatures` ermittelt wurden, läuft diese Funktion einfach über die Artikeltitel, gibt sie aus und zeigt dann die Hauptmerkmale für jeden Artikel an.

Um diese Funktion zu verwenden, laden Sie *newsfeatures.py* neu und rufen sie mit den Titeln und den Ergebnissen aus `showfeatures` auf:

```
>>> reload(newsfeatures)
<module 'newsfeatures' from 'newsfeatures.py'>
>>> newsfeatures.showarticles(artt,topp,pn)
```

Damit wird eine Datei namens *articles.txt* erstellt, die die Artikel mit den drei wichtigsten Merkmalen (mit ihren »stärksten« Wörtern) enthält. Hier ein gutes Beispiel für einen Artikel, der zwei etwa gleich wichtige Merkmale enthält:

```
Attacks in Iraq at record high: Pentagon
5.4890098003 [u'monday', u'said', u'oil', u'iraq', u'attacks', u'two']
5.33447632219 [u'gates', u'iraq', u'pentagon', u'washington', u'over', u'report']
0.618495842404 [u'its', u'iraqi', u'baghdad', u'red', u'crescent', u'monday']
```

Ganz offensichtlich stehen beide Merkmale im Zusammenhang mit dem Irak, sind aber nicht speziell für diesen Artikel, da »oil« oder »gates« nicht erwähnt werden. Durch das Erstellen von Mustern, die in Kombination angewendet werden können, aber nicht direkt auf einen Artikel zugeschnitten sind, kann der Algorithmus viel mehr Artikel mit weniger Merkmalen abdecken.

Hier ein Beispiel eines Artikels mit einem starken einzelnen Merkmal, das nur schlecht auf etwas anderes angewendet werden kann:

```
Yogi Bear Creator Joe Barbera Dies at 95
11.8474089735 [u'barbera', u'team', u'creator', u'hanna', u'dies', u'bear']
2.21373704749 [u'monday', u'said', u'oil', u'iraq', u'attacks', u'two']
0.421760994361 [u'man', u'was', u'year', u'his', u'old', u'kidnapping']
```

Da die Anzahl der verwendeten Muster ziemlich niedrig ist, wird es auch sehr wahrscheinlich viele *Waisen*-Artikel geben, die den anderen Artikeln nur wenig ähneln und keine eigenen Merkmale erhalten haben. Hier ein Beispiel:

```
U.S. Files Charges in Fannie Mae Accounting Case
0.856087848533 [u'man', u'was', u'year', u'his', u'old', u'kidnapping']
0.784659717694 [u'climbers', u'hood', u'have', u'their', u'may', u'deaths']
0.562439763693 [u'will', u'smith', u'news', u'office', u'box', u'all']
```

e-bol.net

Sie sehen, dass die Hauptmerkmale für dieses Beispiel nichts miteinander zu tun haben und eher nach Zufallszuweisung aussehen. Glücklicherweise sind die Gewichtungen hier sehr klein, daher ist es einfach zu erkennen, dass die Merkmale für diesen Artikel alle nicht gut passen.

Daten des Aktienmarkts nutzen

So, wie die NMF gut mit so etwas wie Nominaldaten (den Worthäufigkeiten) umgehen kann, passt sie auch bei Problemen, die echte numerische Daten beinhalten. Dieser Abschnitt wird Ihnen zeigen, wie der gleiche Algorithmus auf das *Handelsvolumen* im US-amerikanischen Aktienmarkt angewendet werden kann, wobei Daten verwendet werden, die von Yahoo! Finance stammen. Diese Daten können eventuell entscheidende Handelstage aufzeigen oder auf Wege hinweisen, wie zugrunde liegende Faktoren die Menge der Aktientransaktionen beeinflussen.

Finanzmärkte werden als eines der wichtigsten Beispiele für kollektive Intelligenz angesehen, da sie eine große Zahl an Teilnehmern besitzen, die unabhängig voneinander vorgehen, unterschiedliche Informationen haben und deren Aktionen nur kleine Ergebnismengen liefern, wie zum Beispiel den Preis und die Menge. Es hat sich herausgestellt, dass es für Individuen ausgesprochen schwierig ist, zukünftige Preise besser vorauszusagen, als es die Gesamtheit der Personen tut. Es gibt umfangreiche akademische Untersuchungen dazu, dass Personengruppen Preise in einem Finanzmarkt besser voraussagen können, als Einzelpersonen dazu in der Lage sind.

Was ist das Handelsvolumen?

Das Handelsvolumen für eine bestimmte Aktie ist die Anzahl an Aktien, die innerhalb eines gewissen Zeitraums (meist eines Tages) gekauft und verkauft werden. Abbildung 10-7 zeigt einen Graphen der Yahoo!-Aktie mit dem Ticker-Symbol *YHOO*. Die obere Linie ist der *Schlusspreis*, also der Preis der letzten Transaktion des Tages. Das Balkendiagramm darunter zeigt das Handelsvolumen.

Sie werden feststellen, dass das Volumen an den Tagen größer zu sein scheint, an denen sich der Aktienpreis stärker ändert. Das passiert häufig dann, wenn ein Unternehmen eine wichtige Mitteilung herausgibt oder die Quartalsergebnisse mitteilt. Spitzen können auch entstehen, wenn es Nachrichten über die Firma oder die Branche gibt. Fehlen solche externen Ereignisse, ist das Handelsvolumen normalerweise, aber nicht immer, für eine gegebene Aktie konstant.

In diesem Abschnitt werden Sie sich das Volumen für eine Reihe von Aktien in einem gewissen Datumsbereich anschauen. Sie können damit nach Änderungsmustern im Volumen suchen, die mehrere Aktien gleichzeitig betrafen, oder nach Ereignissen, die so einflussreich waren, dass sie ihre eigenen Merkmale erhalten. Das

e-bol.net

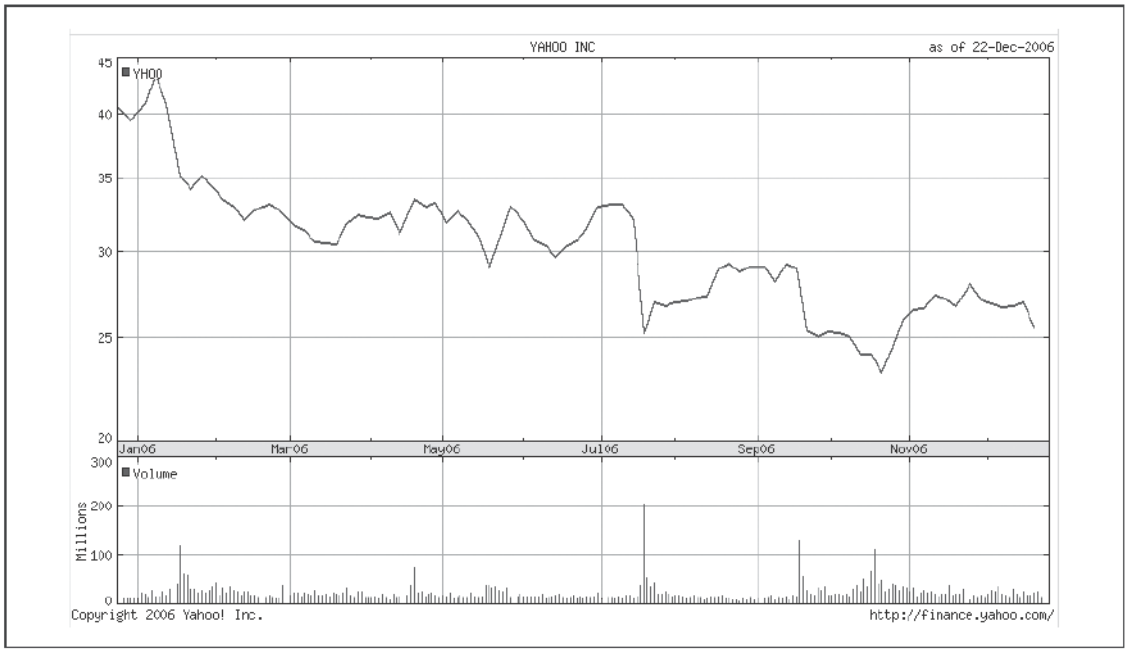

Abbildung 10-7: Aktiengraph mit dem Preis und dem Handelsvolumen

Handelsvolumen wird statt des Schlusspreises genutzt, weil die NMF versucht, positive Merkmale zu finden, die addiert werden können – Preise gehen oft aufgrund von Ereignissen nach unten, die NMF kann aber keine negativen Merkmale finden. Das Volumen lässt sich dagegen leichter modellieren, da es ein »Grundlevel« gibt, das als Reaktion auf äußere Einflüsse ansteigen kann und damit besser in positive Matrizen passt.

Herunterladen der Daten von Yahoo! Finance

Yahoo! Finance ist eine ausgezeichnete Ressource für alle Arten von Finanzdaten, einschließlich Aktienpreisen, Optionen, Wechselkursen und Zinssätzen für Obligationen. Es ermöglicht auch ein bequemes Herunterladen historischer Handelsvolumen und Preisdaten im CSV-Format. Rufen Sie zum Beispiel die URL *http://ichart.finance.yahoo.com/table.csv?s=YHOO&d=11&e=26&f=2006&g=d&a=3&b=12&c=1996&ignore=.csv* auf, können Sie eine Liste der täglichen Aktiendaten in einer kommaseparierten Datei herunterladen. Die ersten paar Zeilen sehen dann so aus:

```
Date,Open,High,Low,Close,Volume,Adj. Close*
22-Dec-06,25.67,25.88,25.45,25.55,14666100,25.55
21-Dec-06,25.71,25.75,25.13,25.48,27050600,25.48
20-Dec-06,26.24,26.31,25.54,25.59,24905600,25.59
19-Dec-06,26.05,26.50,25.91,26.41,18973800,26.41
18-Dec-06,26.89,26.97,26.07,26.30,19431200,26.30
```

Jede Zeile enthält das Datum, den Eröffnungs- und den Schlusskurs, den höchsten und niedrigsten Kurs, das Volumen und den angepassten Schlusspreis. Der ange-

e-bol.net

passte Schlusspreis berücksichtigt noch eventuelle Aktiensplits oder ausgezahlte Dividenden und kann genutzt werden, um genau zu berechnen, wie viel Geld Sie gemacht hätten, wenn Sie die Aktien zwischen zwei verschiedenen Tagen besessen hätten.

In diesem Beispiel werden Sie die Volumendaten für eine Reihe von Aktien laden. Erstellen Sie eine neue Datei namens *stockvolume.py* und fügen Sie den folgenden Code ein, der die kommaseparierten Dateien für eine Reihe von Ticker-Symbolen lädt und in ein Directory steckt. Der Code prüft auch, welcher von ihnen die wenigsten aufgezeichneten Tage enthält. Dieser Wert wird dann als vertikale Größe der Beobachtungs-Matrix genutzt:

```
import nmf
import urllib2
from numpy import *

tickers=['YHOO','AVP','BIIB','BP','CL','CVX',
         'DNA','EXPE','GOOG','PG','XOM','AMGN']

shortest=300
prices={}
dates=None

for t in tickers:
  # Öffnen der URL.
  rows=urllib2.urlopen('http://ichart.finance.yahoo.com/table.csv?'+\
                       's=%s&d=11&e=26&f=2006&g=d&a=3&b=12&c=1996'%t +\
                       '&ignore=.csv').readlines( )


  # Extrahieren des Volumen-Felds aus jeder Zeile.
  prices[t]=[float(r.split(',')[5]) for r in rows[1:] if r.strip( )!='']
  if len(prices[t])<shortest: shortest=len(prices[t])

  if not dates:
    dates=[r.split(',')[0] for r in rows[1:] if r.strip( )!='']
```

Dieser Code öffnet die URL für jedes Ticker-Symbol und lädt die Daten herunter. Dann erstellt er eine Liste, indem er jede Zeile an den Kommata trennt und den Float-Wert des fünften Elements übernimmt, das für diese Aktie das Handelsvolumen enthält.

Vorbereiten einer Matrix

Der nächste Schritt ist das Umwandeln dieses Dictionary in eine Matrix aus Beobachtungen, die an die NMF-Funktion übergeben werden kann. Dafür muss nur eine geschachtelte Liste erstellt werden, in der jede innere Liste das Gesamtvolumen für eine Reihe von Aktien für jeden Tag enthält. Schauen Sie sich zum Beispiel diese Zeilen an:

e-bol.net

```
[[4453500.0, 842400.0, 1396100.0, 1883100.0, 1281900.0,...]
 [5000100.0, 1486900.0, 3317500.0, 2966700.0, 1941200.0,...
 [5222500.0, 1383100.0, 3421500.0, 2885300.0, 1994900.0,...
 [6028700.0, 1477000.0, 8178200.0, 2919600.0, 2061000.0,...]
 ...]
```

Diese Liste zeigt, dass am letzten übermittelten Tag 4453500 Aktien von AMGN gehandelt wurde, 842400 Aktien von AVP und so weiter. Am Tag vorher waren die Zahlen 5000100 und 1486900. Vergleichen Sie das mit dem Nachrichten-Beispiel – die Artikel sind jetzt Tage, die Wörter Aktien und die Wortfrequenzen das Handelsvolumen.

Die Matrix kann mit einer List Comprehension leicht erstellt werden. Die innere Schleife läuft über die Ticker-Liste, während die äußere die Beobachtungsliste (die Tage) abarbeitet. Fügen Sie diesen Code am Ende von *stockvolume.py* hinzu:

```
l1=[[prices[tickers[i]][j]
     for i in range(len(tickers))]
    for j in range(shortest)]
```

Starten der NMF

Jetzt müssen Sie nur noch die Funktion `factorize` aus dem Modul `nmf` aufrufen. Geben Sie die Anzahl der zu suchenden Merkmale an. Bei dieser Menge an Aktien sind vier oder fünf eine gute Wahl.

Fügen Sie diesen Code am Ende von *stockvolume.py* an:

```
w,h= nmf.factorize(matrix(l1),pc=5)

print h
print w
```

Sie können das Ganze nun von der Befehlszeile aus starten, um zu sehen, ob es funktioniert:

```
$ python stockvolume.py
```

Die angezeigten Matrizen repräsentieren die Gewichtungen und die Merkmale. Jede Zeile in der Merkmals-Matrix ist ein Merkmal, das wiederum für eine Reihe von Handelsvolumen für die Aktien steht, das mit anderen Merkmalen summiert werden kann, um das Handelsvolumen für diesen Tag neu zu berechnen. Jede Zeile in der Gewichtungs-Matrix steht für einen bestimmten Tag, und die Werte geben an, wie stark jedes der Merkmale an diesem Tag gültig ist.

Anzeigen der Ergebnisse

Es ist offensichtlich recht schwierig, die Matrizen direkt zu interpretieren, daher brauchen Sie noch etwas Code, um die Merkmale besser anzeigen zu können. Inter-

e-bol.net

essant wären die Volumenbeiträge jedes Merkmals für jede Aktie, aber auch die Tage, die am meisten mit diesen Merkmalen verknüpft sind.

Fügen Sie diesen Code am Ende von *stockvolume.py* hinzu:

```
# Schleife über alle Merkmale.
for i in range(shape(h)[0]):
  print "Merkmal %d" %i

  # Wichtigste Aktien für dieses Merkmal ermitteln.
  ol=[(h[i,j],tickers[j]) for j in range(shape(h)[1])]
  ol.sort()
  ol.reverse()
  for j in range(12):
    print ol[j]
  print

  # Wichtigste Tage für dieses Merkmal anzeigen.
  porder=[(w[d,i],d) for d in range(300)]
  porder.sort()
  porder.reverse()
  print [(p[0],dates[p[1]]) for p in porder[0:3]]
  print
```

Da so eine Menge Text entsteht, ist es vermutlich am besten, die Ausgabe in eine Datei umzuleiten. Geben Sie an der Befehlszeile Folgendes ein:

```
$ python stockvolume.py > stockfeatures.txt
```

Die Datei *stockfeatures.txt* enthält nun eine Liste mit Merkmalen, in der auch die Aktien aufgeführt sind, die am meisten von ihnen beeinflusst werden, sowie die Tage, an denen die Merkmale am stärksten wirken. Hier ein Beispiel, das aus der Datei ausgewählt wurde, weil es eine sehr hohe Gewichtung für eine bestimmte Aktie an einem bestimmten Tag zeigt:

```
Merkmal 4
(74524113.213559602, 'YHOO')
(6165711.6749675209, 'GOOG')
(5539688.0538382991, 'XOM')
(2537144.3952459987, 'CVX')
(1283794.0604679288, 'PG')
(1160743.3352889531, 'BP')
(1040776.8531969623, 'AVP')
(811575.28223116993, 'BIIB')
(679243.76923785623, 'DNA')
(377356.4897763988, 'CL')
(353682.37800343882, 'EXPE')
(0.31345784102699459, 'AMGN')

[(7.950090052903934, '19-Jul-06'),
 (4.7278341805021329, '19-Sep-06'),
 (4.6049947721971245, '18-Jan-06')]
```

e-bol.net

Sie können sehen, dass dieses Merkmal fast ausschließlich für YHOO wirkt und am 19. Juli 2006 sehr stark ist. Wie der Zufall es will, gab es an diesem Tag ein besonders großes Handelsvolumen für Yahoo!, was zu einer Earning Guidance führte.

Ein anderes Merkmal, das sich eher auf mehr als eine Firma auswirkte, ist dieses:

```
Merkmal 2
(46151801.813632453, 'GOOG')
(24298994.720555616, 'YHOO')
(10606419.91092159, 'PG')
(7711296.6887903402, 'CVX')
(4711899.0067871698, 'BIIB')
(4423180.7694432881, 'XOM')
(3430492.5096612777, 'DNA')
(2882726.8877627672, 'EXPE')
(2232928.7181202639, 'CL')
(2043732.4392455407, 'AVP')
(1934010.2697886101, 'BP')
(1801256.8664912341, 'AMGN')

[(2.9757765047938824, '20-Jan-06'),
 (2.8627791325829448, '28-Feb-06'),
 (2.356157903021133, '31-Mar-06'),
```

Dieses Merkmal steht für große Handelsspitzen bei Google, was an den drei Haupttagen an neuen Bekanntgaben lag. Der 20. Januar als stärkster Tag war der Tag, an dem Google bekannt gab, dass es keine Informationen über ihre Such-Engine an die amerikanische Regierung weitergeben würde. Wirklich interessant an diesem Merkmal ist, dass Ereignisse, die das Handelsvolumen von Google beeinflussen, sich auch stark auf das Handelsvolumen von Yahoo! auszuwirken scheinen, selbst wenn sie nur wenig mit Yahoo! zu tun haben. Das zweite Datum in der Liste ist ein Tag mit großem Handelsvolumen, an dem Googles Finanzchef mitteilte, dass das Wachstum sich verlangsamen würde – die Charts zeigen, dass auch bei Yahoo! das Handelsvolumen größer war als sonst. Vermutlich haben die Leute überlegt, wie die Information Yahoo! beeinflussen würde.

Es ist wichtig, sich des Unterschieds bewusst zu sein zwischen dem, was hier gezeigt wird, und dem einfachen Finden von Zusammenhängen zwischen Handelsvolumen von Aktien. Die oben gezeigten zwei Merkmale zeigen, dass es Tage gibt, an denen Google und Yahoo! ähnliche Volumenmuster aufzeigen, während sie an anderen Tagen völlig verschieden sind. Schaut man sich nur die Korrelationen an, würden alle diese Beziehungen gleich gewertet werden, und es würde nicht berücksichtigt, dass es einfach ein paar Tage gibt, an denen Yahoo! Bekanntmachungen gemacht hat, die einen großen Einfluss hatten.

Dieses Beispiel nutzt nur eine kleine Auswahl an Aktien, um einen Punkt deutlich zu machen, aber mit einer größeren Auswahl an Aktien und der Suche nach mehr Merkmalen würden noch komplexere Interaktionen aufgedeckt werden.

e-bol.net

Übungen

1. *Andere Nachrichtenquellen*. Das Beispiel in diesem Kapitel hat hauptsächlich reine Nachrichtenquellen genutzt. Versuchen Sie, ein paar der bekannten politischen Blogs hinzuzufügen (*http://technorati.com* ist da ein guter Ausgangspunkt für eine Suche). Wie wirkt sich das auf die Ergebnisse aus? Gibt es Merkmale, die vor allem für politische Kommentare zutreffen? Gibt es Nachrichten, die mit den zugehörigen Kommentaren gruppiert werden?
2. *K-Means-Clusteranalyse*. Für die Artikel-Matrix wurde eine hierarchische Clusteranalyse durchgeführt, aber was passiert, wenn Sie eine K-Means-Clusteranalyse durchführen? Wie viele Cluster brauchen Sie, um die verschiedenen Artikel gut trennen zu können? Wie steht das mit der Anzahl an Merkmalen in Zusammenhang, die Sie brauchen, um alle Themen zu extrahieren?
3. *Optimieren der Faktorisierung*. Können Sie den Optimierungscode aus Kapitel 5 nutzen, um die Matrix zu faktorisieren? Ist dies schneller oder langsamer? Wie sehen die Ergebnisse im Vergleich zum hier genutzten Algorithmus aus?
4. *Abbruch-Kriterien*. Der NMF-Algorithmus in diesem Kapitel bricht ab, wenn die Kosten bei 0 liegen oder wenn die maximale Anzahl an Iterationen erreicht ist. Manchmal gibt es so gut wie keine Verbesserung mehr, wenn schon eine sehr gute, wenn auch nicht perfekte Lösung erreicht wurde. Passen Sie den Code so an, dass er abbricht, wenn sich die Kosten pro Iteration nicht um mindestens 1 % verbessern.
5. *Alternative Anzeigemethoden*. Die Funktionen, die in diesem Kapitel die Ergebnisse ausgeben, sind einfach und zeigen wichtige Merkmale, aber es gehen auch eine Menge Zusammenhänge verloren. Können Sie sich andere Arten der Anzeige von Ergebnissen vorstellen? Versuchen Sie, eine Funktion zu schreiben, die den Artikel vollständig anzeigt und die Schlüsselwörter jedes Merkmals hervorhebt oder die vielleicht einen Chart mit dem Handelsvolumen ausgibt, bei dem die wichtigen Tage hervorgehoben sind.

e-bol.net

KAPITEL 11

Entwickeln von Intelligenz

In diesem Buch haben Sie viele verschiedene Probleme kennengelernt und jedes Mal einen Algorithmus verwendet, der darauf ausgerichtet war, das spezielle Problem zu lösen. In manchen der Beispiele mussten Sie an den Parametern drehen oder die Optimierung zu Hilfe nehmen, um gute Parameter zu finden. Dieses Kapitel wird anders an Probleme herangehen. Anstatt einen Algorithmus zu wählen, der auf ein Problem angewendet werden soll, werden Sie ein Programm bauen, das versucht, automatisch das beste Programm zum Lösen eines Problems zu kreieren. Im Prinzip werden Sie also einen Algorithmus erstellen, der einen Algorithmus erstellt.

Dazu werden Sie eine Technik namens *genetische Programmierung* verwenden. Da dies das letzte Kapitel ist, in dem Sie einen vollständig neuen Algorithmus kennenlernen werden, habe ich einen Bereich herausgesucht, der neu, spannend und immer noch Bestandteil der aktuellen Forschung ist. Dieses Kapitel ist ein wenig anders als die anderen, da es weder offene APIs noch frei verfügbare Datenmengen nutzt und weil Programme, die sich selbst aufgrund ihrer Interaktion mit vielen Personen modifizieren können, eine interessante und andere Form der kollektiven Intelligenz sind. Genetische Programmierung ist ein sehr großes Gebiet, über das viele Bücher geschrieben wurden, daher werden Sie hier nur eine Einführung erhalten, aber ich hoffe, dass sie ausreicht, um Ihnen die Möglichkeiten schmackhaft zu machen und Sie vielleicht sogar dazu zu bringen, selbst Forschungen anzustellen und zu experimentieren.

Bei den zwei Problemen in diesem Kapitel geht es um das Nachbauen einer mathematischen Funktion für gegebene Daten und um das automatische Erstellen eines KI-Spielers (KI = Künstliche Intelligenz, im Englischen AI = Artificial Intelligence) für ein einfaches Brettspiel. Dies ist nur ein sehr kleiner Ausschnitt aus den Möglichkeiten der genetischen Programmierung – im Prinzip ist die Rechenleistung das Einzige, was die Art von Problemen einschränkt, die mit genetischer Programmierung gelöst werden können.

e-bol.net

Was ist genetische Programmierung?

Genetische Programmierung ist eine Technik des maschinellen Lernens, die von der Theorie der biologischen Evolution angeregt wurde. Sie funktioniert im Allgemeinen so, dass man mit einer großen Zahl von Programmen beginnt (als *Population* bezeichnet), die entweder zufällig erzeugt wurden oder von Hand angelegt und als halbwegs gute Lösungen bekannt sind. Die Programme werden dann als Konkurrenten auf eine benutzerdefinierte Aufgabe losgelassen. Dabei kann es sich um ein Spiel handeln, in dem die Programme direkt gegeneinander antreten, oder um einen individuellen Test, um herauszufinden, welche Programme bessere Ergebnisse liefern. Nach dem Vergleich können die Programme in einer Liste angeordnet werden – von den besten bis zu den schlechtesten.

Als Nächstes – und hier kommt die Evolution ins Spiel – werden die besten Programme repliziert und auf zwei verschiedene Arten verändert. Die einfachere Form ist die *Mutation*, bei der bestimmte Teile des Programms in zufälliger Weise ein wenig verändert werden – in der Hoffnung, dass eine gute Lösung dadurch noch besser wird. Die andere Form ist ein *Crossover* (Kreuzung, manchmal auch als *Breeding* bzw. *Zucht* bezeichnet), bei dem ein Teil eines der besten Programme genommen wird und durch einen Teil eines der anderen besten Programme ersetzt wird. Diese Replikations- und Modifikationsprozedur sorgt für viele neue Programme, die zwar auf den besten Programmen basieren, sich aber doch von ihnen unterscheiden.

In jeder Runde wird die Qualität des Programms mit einer *Fitnessfunktion* überprüft. Da die Größe der Population konstant gehalten wird, werden viele der *schlechtesten* Programme aus der Population entfernt, um Platz für die neuen Programme zu schaffen. Die neue Population wird als »nächste Generation« bezeichnet, und die gesamte Prozedur wird wiederholt. Da die besten Programme beibehalten und modifiziert werden, wird davon ausgegangen, dass sie mit jeder Generation besser und besser werden – ähnlich wie die Teenager, die immer schlauer werden als ihre Eltern.

Neue Generationen werden so lange erzeugt, bis eine Abbruchbedingung erreicht wird, die je nach Problem so aussehen kann:

- Die perfekte Lösung ist gefunden.
- Es wurde eine Lösung gefunden, die gut genug ist.
- Die Lösung hat sich in vielen Generationen nicht verbessert.
- Die Anzahl der Generationen hat eine bestimmte Grenze erreicht.

Bei manchen Problemen, wie dem Bestimmen einer mathematischen Funktion, die eine Reihe von Eingaben korrekt auf eine Ausgabe abbildet, ist eine perfekte Lösung möglich. Bei anderen, wie zum Beispiel einem Brettspiel, kann es sein, dass es keine

e-bol.net

perfekte Lösung gibt, da die Qualität einer Lösung von der Strategie des Gegners abhängt.

Sie erhalten einen Überblick über den Ablauf der genetischen Programmierung in Abbildung 11-1.

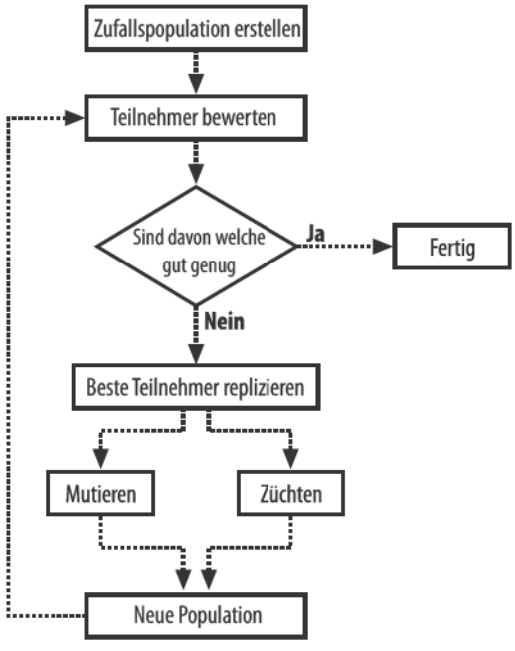

Abbildung 11-1: Ablauf der genetischen Programmierung

Erfolg der genetischen Programmierung

Die genetische Programmierung existiert ungefähr seit den 1980er-Jahren, ist aber sehr rechenintensiv. Aufgrund der damals vorhandenen Rechenleistung konnte sie für nicht viel mehr als einfache Probleme genutzt werden. Durch die immer schneller werdenden Computer ließ sich die genetische Programmierung auf immer komplexere Probleme anwenden. Viele schon patentierte Erfindungen wurden durch genetische Programmierung neu erfunden oder verbessert, und diverse neue patentwürdige Erfindungen entstanden durch Computer.

Die Techniken der genetischen Programmierung wurden beim Entwurf von Antennen für die NASA angewendet, ebenso bei photonischen Kristallen, in der Optik, bei Quantencomputersystemen und in anderen wissenschaftlichen Bereichen. Auch bei der Entwicklung von Programmen für viele Spiele, wie zum Beispiel Schach und Backgammon, fand die Programmierung Einzug. 1998 stellten Forscher der Carnegie Mellon University ein Roboterteam auf, das vollständig durch genetische Programme gesteuert wurde und am RoboCup-Fußballwettbewerb teilnahm. Dabei erreichte man einen Platz im Mittelfeld.

e-bol.net

Genetische Programmierung versus genetische Algorithmen

In Kapitel 5 wurde eine Reihe von Algorithmen vorgestellt, die als *genetische Algorithmen* bekannt sind. Genetische Algorithmen sind eine Optimierungstechnik, die die Idee des Evolutionsdrucks nutzt, um das beste Ergebnis auszuwählen. Bei jeder Form der Optimierung haben Sie schon einen Algorithmus oder eine Metrik gewählt und versuchen nur noch, die besten Parameter dafür zu finden.

Wie die Optimierung braucht die genetische Programmierung eine Möglichkeit zu messen, wie gut eine Lösung ist – aber anders als die Optimierung sind die Lösungen nicht nur eine Reihe von Parametern, die auf einen gegebenen Algorithmus angewendet werden. Stattdessen werden der Algorithmus selbst und alle seine Parameter automatisch durch den Evolutionsdruck entworfen.

Programme als Bäume

Um Programme zu erstellen, die sich testen, mutieren und züchten lassen, brauchen Sie eine Möglichkeit, sie in Ihrem Python-Code zu repräsentieren und auszuführen. Die Repräsentation muss sich leicht anpassen lassen können, und – was noch wichtiger ist – sie muss auf jeden Fall ein echtes Programm sein. Das Generieren von Zufalls-Strings, die dann als Python-Code behandelt werden sollen, geht also nicht. Forscher haben ein paar Möglichkeiten entwickelt, Programme für die genetische Programmierung zu repräsentieren, wobei die am meisten genutzte eine Baumrepräsentation ist.

Der Code der meisten Programmiersprachen wird, wenn man ihn kompiliert oder interpretiert, zunächst in einen *Syntaxbaum* (*Parse Tree*) umgewandelt, der dem sehr ähnelt, mit dem Sie hier arbeiten werden. (Die Programmiersprache Lisp und ihre Ableger sind Beispiele dafür, wie man einen Syntaxbaum direkt eingeben kann.) Ein Beispiel für einen Syntaxbaum sehen Sie in Abbildung 11-2.

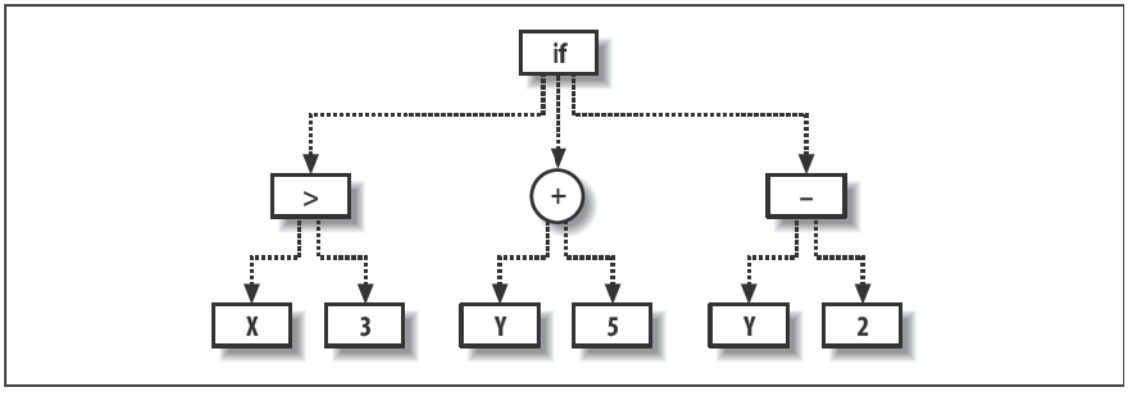

Abbildung 11-2: Programmbaum-Beispiel

e-bol.net

Jeder Knoten steht entweder für eine Operation auf seinen Kindknoten oder für einen Endpunkt, zum Beispiel einen Parameter mit einem konstanten Wert. So ist zum Beispiel der runde Knoten eine Summenoperation auf seinen beiden Zweigen – in diesem Fall den Werten Y und 5. Sobald dieser Punkt ausgewertet ist, wird er an den darüber liegenden Knoten übergeben, der dann seine eigenen Operationen auf seinen Zweigen ausführt. Sie werden bemerkt haben, dass einer der Knoten die Operation »if« hat. Das bedeutet, dass geprüft wird, ob der ganz linke Zweig zu true ausgewertet werden kann. In dem Fall wird der mittlere Zweig zurückgegeben, ansonsten der Zweig ganz rechts.

Arbeitet man den kompletten Baum ab, sieht man, dass er der folgenden Python-Funktion entspricht:

```
def func(x,y)
  if x>3:
    return y + 5
  else:
    return y - 2
```

Auf den ersten Blick scheint es so zu sein, dass diese Bäume nur zum Aufbau sehr einfacher Funktionen taugen. Man muss hier zwei Dinge berücksichtigen. Zum einen können die Knoten, die den Baum ergeben, potenziell sehr komplexe Funktionen darstellen, wie zum Beispiel Entfernungsmessungen oder Gauß-Berechnungen. Zum anderen können Bäume rekursiv gemacht werden, indem man auf Knoten verweist, die sich weiter oben im Baum befinden. Das Erstellen solcher Bäume ermöglicht Schleifen und andere, kompliziertere Kontrollstrukturen.

Darstellen von Bäumen in Python

Jetzt können Sie Baumprogramme in Python erstellen. Die Bäume bestehen aus Knoten, die abhängig von den mit ihnen verbundenen Funktionen eine gewissen Zahl an Kindknoten haben. Manche der Knoten werden Parameter zurückgeben, die dem Programm übergeben wurden, andere Konstanten, und die interessanteren werden Operationen mit ihren Kindknoten anstellen.

Erstellen Sie eine neue Datei namens *gp.py* und legen Sie vier neue Klassen mit den Namen `fwrapper`, `node`, `paramnode` und `constnode` an:

```
from random import random,randint,choice
from copy import deepcopy
from math import log

class fwrapper:
  def __init__(self,function,childcount,name):
    self.function=function
    self.childcount=childcount
    self.name=name
```

e-bol.net

```
class node:
  def __init__(self,fw,children):
    self.function=fw.function
    self.name=fw.name
    self.children=children

  def evaluate(self,inp):
    results=[n.evaluate(inp) for n in self.children]
    return self.function(results)

class paramnode:
  def __init__(self,idx):
    self.idx=idx

  def evaluate(self,inp):
    return inp[self.idx]

class constnode:
  def __init__(self,v):
    self.v=v
  def evaluate(self,inp):
    return self.v
```

Bei den Klassen handelt es sich um folgende:

`fwrapper`
: Ein Wrapper für die Funktionen, die von den Funktionsknoten verwendet werden. Seine Elementvariablen sind der Name der Funktion, die Funktion selbst und die Anzahl an Parametern, die sie erwartet.

`node`
: Die Klasse für Funktionsknoten (also Knoten mit Kindern). Diese wird mit einem `fwrapper` initialisiert. Wenn `evaluate` aufgerufen wird, werden die Kindknoten ausgewertet, und die Funktion wird auf ihre Ergebnisse angewandt.

`paramnode`
: Die Klasse für Knoten, die nur einen der Parameter zurückgeben, die das Programm erhalten hat. Seine Methode `evaluate` liefert den durch `idx` angegebenen Parameter zurück.

`constnode`
: Knoten, die einen konstanten Wert zurückgeben. Die Methode `evaluate` liefert einfach den Wert zurück, mit dem die Klasse initialisiert wurde.

Sie werden auch ein paar Funktionen haben wollen, die die Knoten anwenden können. Dazu müssen Sie Funktionen erstellen und ihnen dann für die Verwendung mit `fwrapper` Namen und die Parameteranzahl mitgeben. Fügen Sie diese Liste mit Funktionen der Datei *gp.py* hinzu:

```
addw=fwrapper(lambda l:l[0]+l[1],2,'add')
subw=fwrapper(lambda l:l[0]-l[1],2,'subtract')
mulw=fwrapper(lambda l:l[0]*l[1],2,'multiply')
```

e-bol.net

```
def iffunc(l):
  if l[0]>0: return l[1]
  else: return l[2]
ifw=fwrapper(iffunc,3,'if')

def isgreater(l):
  if l[0]>l[1]: return 1
  else: return 0
gtw=fwrapper(isgreater,2,'isgreater')

flist=[addw,mulw,ifw,gtw,subw]
```

Manche der einfacheren Funktionen wie `add` und `subtract` können mithilfe von `lambda` inline definiert werden, bei anderen müssen Sie die Funktion in einem eigenen Block definieren. Auf jeden Fall werden sie dann mit ihren Namen und der Anzahl der Parameter in einem `fwrapper` eingeschlossen. Die letzte Zeile erstellt eine Liste all dieser Funktionen, sodass sie später einfach zufällig ausgewählt werden können.

Aufbauen und Auswerten von Bäumen

Sie können jetzt den in Abbildung 11-2 gezeigten Baum mithilfe der gerade erstellten Klasse `node` aufbauen. Fügen Sie die Funktion `exampletree` der Datei *gp.py* hinzu, um den Baum zu erstellen:

```
def exampletree( ):
  return node(ifw,[
                  node(gtw,[paramnode(0),constnode(3)]),
                  node(addw,[paramnode(1),constnode(5)]),
                  node(subw,[paramnode(1),constnode(2)]),
                  ]
                )
```

Starten Sie eine Python-Session, um Ihr Programm auszuprobieren:

```
>>> import gp
>>> exampletree=gp.exampletree( )
>>> exampletree.evaluate([2,3])
1
>>> exampletree.evaluate([5,3])
8
```

Das Programm führt erfolgreich die gleiche Funktion wie der äquivalente Codeblock aus, also haben Sie es geschafft, eine kleine, baumbasierte Sprache und einen Interpreter in Python aufzubauen. Diese Sprache lässt sich leicht mit weiteren Knotentypen erweitern und wird als Grundlage für das Verstehen genetischer Programmierung in diesem Kapitel dienen. Versuchen Sie, ein paar andere einfache Programmbäume aufzubauen, um sicherzustellen, dass Sie sie verstanden haben.

e-bol.net

Anzeigen des Programms

Da Sie Programmbäume automatisch erstellen werden und nicht wissen, wie ihre Struktur aussieht, ist es wichtig, eine Möglichkeit zu haben, sie so anzuzeigen, dass sie leicht interpretiert werden können. Glücklicherweise ist die Klasse `node` so entworfen, dass jeder Knoten eine String-Repräsentation mit dem Namen seiner Funktion besitzt, sodass eine Anzeigefunktion nur diesen String zurückgeben und die Strings der Kindknoten ausgeben lassen muss. Um das Ganze leichter lesbar zu gestalten, sollten die Kindknoten zudem eingerückt sein, damit Sie die Eltern-Kind-Beziehung im Baum besser erkennen können.

Erstellen Sie eine neue Methode in der Klasse `node` mit dem Namen `display`, die einen String ausgibt, der den Baum repräsentiert:

```
def display(self,indent=0):
  print (' '*indent)+self.name
  for c in self.children:
    c.display(indent+1)
```

Sie werden auch eine Methode `display` für die Klasse `paramnode` anlegen müssen, die einfach den Index des Parameters ausgibt, den sie liefert:

```
def display(self,indent=0):
  print '%sp%d' % (' '*indent,self.idx)
```

Und schließlich eine Methode für die Klasse `constnode`:

```
def display(self,indent=0):
  print '%s%d' % (' '*indent,self.v)
```

Nutzen Sie diese Methoden, um den Baum auszugeben:

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> exampletree=gp.exampletree()
>>> exampletree.display()
if
 isgreater
  p0
  3
 add
  p1
  5
 subtract
  p1
  2
```

Wenn Sie Kapitel 7 gelesen haben, werden Sie erkennen, dass die Ausgabe sehr derjenigen ähnelt, die dort für die Entscheidungsbäume genutzt wurde. Kapitel 7 zeigt auch, wie man solche Bäume grafisch ausgeben kann, um eine übersichtlichere Anzeige zu erhalten. Wenn Sie möchten, können Sie die gleichen Ideen nutzen, um eine grafische Ausgabe Ihres Baumprogramms zu erzeugen.

e-bol.net

Erzeugen der initialen Population

Obwohl es möglich ist, Programme für die genetische Programmierung per Hand zu erzeugen, besteht die erste Population in einem Großteil der Fälle aus Zufallsprogrammen. Dadurch ist der Einstieg leichter, da es nicht notwendig ist, viele Programme zu erzeugen, die ein Problem nur fast lösen. Außerdem entsteht so eine größere *Diversität* in der initialen Population – viele Programme, die aber nur von einem einzelnen Programmierer erstellt wurden, um ein Problem zu lösen, sind sehr wahrscheinlich sehr ähnlich, und auch wenn die von ihnen gegebenen Antworten fast richtig sein mögen, kann es sein, dass die ideale Lösung ganz anders aussieht. Sie werden später noch mehr darüber lernen, wie wichtig Diversität ist.

Um ein zufälliges Programm zu erzeugen, erstellt man einen Wurzelknoten mit einer zufällig ausgewählten Funktion und legt dann so viele Zufalls-Kindknoten an wie notwendig, die wiederum ihre eigenen zufällig zugewiesenen Kindknoten haben können. Wie die meisten Funktionen, die mit Bäumen arbeiten, lässt sich das am besten rekursiv definieren. Fügen Sie eine neue Funktion `makerandomtree` der Datei *gp.py* hinzu:

```
def makerandomtree(pc,maxdepth=4,fpr=0.5,ppr=0.6):
  if random( )<fpr and maxdepth>0:
    f=choice(flist)
    children=[makerandomtree(pc,maxdepth-1,fpr,ppr)
              for i in range(f.childcount)]
    return node(f,children)
  elif random( )<ppr:
    return paramnode(randint(0,pc-1))
  else:
    return constnode(randint(0,10))
```

Diese Funktion erzeugt einen Knoten mit einer zufälligen Funktion und prüft dann, wie viele Kindknoten für diese Funktion notwendig sind. Für jeden erforderlichen Kindknoten ruft sich die Funktion selbst auf, um einen neuen Knoten zu erzeugen. Auf diese Art und Weise wird ein ganzer Baum erstellt – mit Zweigen, die erst dann enden, wenn die Funktion keine weiteren Kindknoten braucht (weil sie eine Konstante oder eine Eingabevariable zurückliefert). Der Parameter `pc`, der im gesamten Kapitel genutzt wird, enthält die Anzahl an Parametern, die der Baum übergeben bekommt. Der Parameter `fpr` gibt die Wahrscheinlichkeit an, dass der neu erstellte Knoten ein Funktionsknoten sein wird, während `ppr` die Wahrscheinlichkeit festlegt, dass es ein Parameterknoten wird, falls es kein Funktionsknoten ist.

Probieren Sie diese Funktion in Ihrer Python-Session aus, um ein paar Programme zu bauen und um zu sehen, was für Ergebnisse Sie mit unterschiedlichen Variablen erhalten:

```
>>> random1=gp.makerandomtree(2)
>>> random1.evaluate([7,1])
7
>>> random1.evaluate([2,4])
```

e-bol.net

```
2
>>> random2=gp.makerandomtree(2)
>>> random2.evaluate([5,3])
1
>>> random2.evaluate([5,20])
0
```

Wenn alle Endknoten des Programms Konstanten zurückgeben, wird das Programm auf die Eingabeparameter keine Rücksicht nehmen und das Ergebnis immer unabhängig davon sein. Sie können die im vorigen Abschnitt definierte Funktion nutzen, um die zufällig erzeugten Bäume auszugeben:

```
>>> random1.display()
p0
>>> random2.display()
subtract
 7
 multiply
  isgreater
   p0
   p1
  if
   multiply
    p1
    p1
   p0
   2
```

Sie werden feststellen, dass einige der Bäume ziemlich tief sind, da jeder Zweig wächst, bis er einen Knoten ohne Kinder erreicht. Das ist der Grund dafür, dass es wichtig ist, eine maximale Tiefe vorzugeben, denn sonst können die Bäume sehr groß werden und potenziell den Stack überlaufen lassen.

Testen einer Lösung

Sie würden nun alles haben, was Sie brauchen, um Programme automatisch erstellen zu lassen, wenn Sie nur so lange Zufallsprogramme erstelten, bis eines richtig ist. Das wäre aber offenkundig ausgesprochen unpraktisch, da es unendlich viele mögliche Programme gibt und es sehr unwahrscheinlich ist, dass Sie in einem vernünftigen Zeitrahmen zufällig über ein korrektes stolpern. Hier ist es nun an der Zeit, sich erst einmal einen Weg zu überlegen, wie man eine Lösung auf Korrektheit testet und wie weit sie im Fall, dass sie nicht korrekt ist, von der richtigen Lösung abweicht.

Ein einfacher mathematischer Test

Einer der einfachsten Tests für die genetische Programmierung ist die Rekonstruktion einer einfachen mathematischen Funktion. Stellen Sie sich vor, Sie hätten eine Tabelle mit Eingabewerten und einem Ausgabewert, die so aussieht wie Tabelle 11-1.

e-bol.net

Tabelle 11-1: Daten und Ergebnisse für eine unbekannte Funktion

X	Y	Ergebnis
26	35	829
8	24	141
20	1	467
33	11	1215
37	16	1517

Es gibt also eine Funktion, die X und Y auf das Ergebnis abbildet, aber Sie wissen nicht, wie sie aussieht. Ein Statistiker würde sich diese Tabelle vielleicht vornehmen und eine Regressionsanalyse darüberlaufen lassen, aber dafür bräuchte man schon eine Vermutung zur Struktur der Formel. Eine andere Möglichkeit ist, ein Schätzmodell mit den k-nächsten Nachbarn zu bauen, wie Sie es in Kapitel 8 getan haben. Dafür müssten Sie aber alle Daten aufbewahren. In manchen Fällen benötigen Sie nur eine Formel, vielleicht um sie in einem anderen viel einfacheren Programm einzubauen oder um anderen Personen zu beschreiben, was hier geschieht.

Ich bin sicher, dass Sie es vor Spannung kaum aushalten, daher werde ich Ihnen erzählen, welche Funktion dies ist. Fügen Sie `hiddenfunction` der Datei *gp.py* hinzu:

```
def hiddenfunction(x,y):
    return x**2+2*y+3*x+5
```

Sie werden diese Funktion nutzen, um eine Datenmenge aufzubauen, mit der Sie Ihre generierten Programme testen können. Fügen Sie eine neue Funktion `buildhiddenset` hinzu, die die Datenmenge erstellt:

```
def buildhiddenset( ):
  rows=[]
  for i in range(200):
    x=randint(0,40)
    y=randint(0,40)
    rows.append([x,y,hiddenfunction(x,y)])
  return rows
```

Und nutzen Sie nun die Funktion, um eine Datenmenge in Ihrer Python-Session anzulegen:

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> hiddenset=gp.buildhiddenset( )
```

Natürlich wissen Sie, wie die Funktion aussieht, die die Daten erzeugt hat, aber der echte Test ist ja, ob die genetische Programmierung sie reproduzieren kann, ohne es zu wissen.

Erfolg messen

Wie bei der Optimierung ist es notwendig, messen zu können, wie gut eine Lösung ist. In diesem Fall testen Sie ein Programm gegen ein numerisches Ergebnis, daher

e-bol.net

lässt sich ein Programm recht einfach prüfen, indem Sie schauen, wie nahe es der korrekten Antwort kommt. Fügen Sie `scorefunction` *gp.py* hinzu:

```
def scorefunction(tree,s):
  dif=0
  for data in s:
    v=tree.evaluate([data[0],data[1]])
    dif+=abs(v-data[2])
  return dif
```

Diese Funktion prüft jede Zeile in der Datenmenge, berechnet das Ergebnis der Funktion und vergleicht es mit dem echten Ergebnis. Dann addiert sie alle Differenzen, wobei niedrigere Werte für bessere Programme stehen – ein Rückgabewert von 0 zeigt, dass das Programm jedes Ergebnis korrekt ermittelt hat. Sie können nun ein paar Ihrer generierten Programme in Ihrer Python-Session prüfen, um zu sehen, wie gut sie sich machen:

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> gp.scorefunction(random2,hiddenset)
137646
>>> gp.scorefunction(random1,hiddenset)
125489
```

Da Sie nur ein paar Programme generiert haben und diese komplett zufällig entstanden sind, ist die Chance verschwindend gering, dass eines von ihnen schon die korrekte Funktion darstellt. (Sollte eines Ihrer Programme trotzdem bereits die korrekte Funktion sein, schlage ich vor, dass Sie dieses Buch beiseite legen und einen Lottoschein ausfüllen.) Auf jeden Fall haben Sie jetzt eine Möglichkeit zu prüfen, wie gut ein Programm bei der Vorhersage einer mathematischen Funktion arbeitet. Das ist wichtig für die Entscheidung darüber, welche Programme es in die nächste Generation schaffen.

Mutieren von Programmen

Nachdem die besten Programme ausgewählt sind, werden sie für die nächste Generation repliziert und modifiziert. Wie schon erwähnt, nimmt die Mutation ein einzelnes Programm und passt es ein wenig an. Die Baumprogramme können auf unterschiedlichste Art und Weise angepasst werden – indem man die Funktion eines Knotens ändert oder seine Zweige anpasst. Eine geänderte Funktion, die zu einer anderen Zahl erforderlicher Kindknoten führt, löscht entweder Zweige oder fügt neue hinzu, wie dies in Abbildung 11-3 zu sehen ist.

Eine andere Mutationsmöglichkeit ist das Ersetzen eines Unterbaums durch einen vollständig neuen, zu sehen in Abbildung 11-4.

Es sollte nicht allzu viel mutiert werden. Sie sollten zum Beispiel nicht die Mehrheit der Knoten in einem Baum verändern. Stattdessen können Sie eine recht geringe

e-bol.net

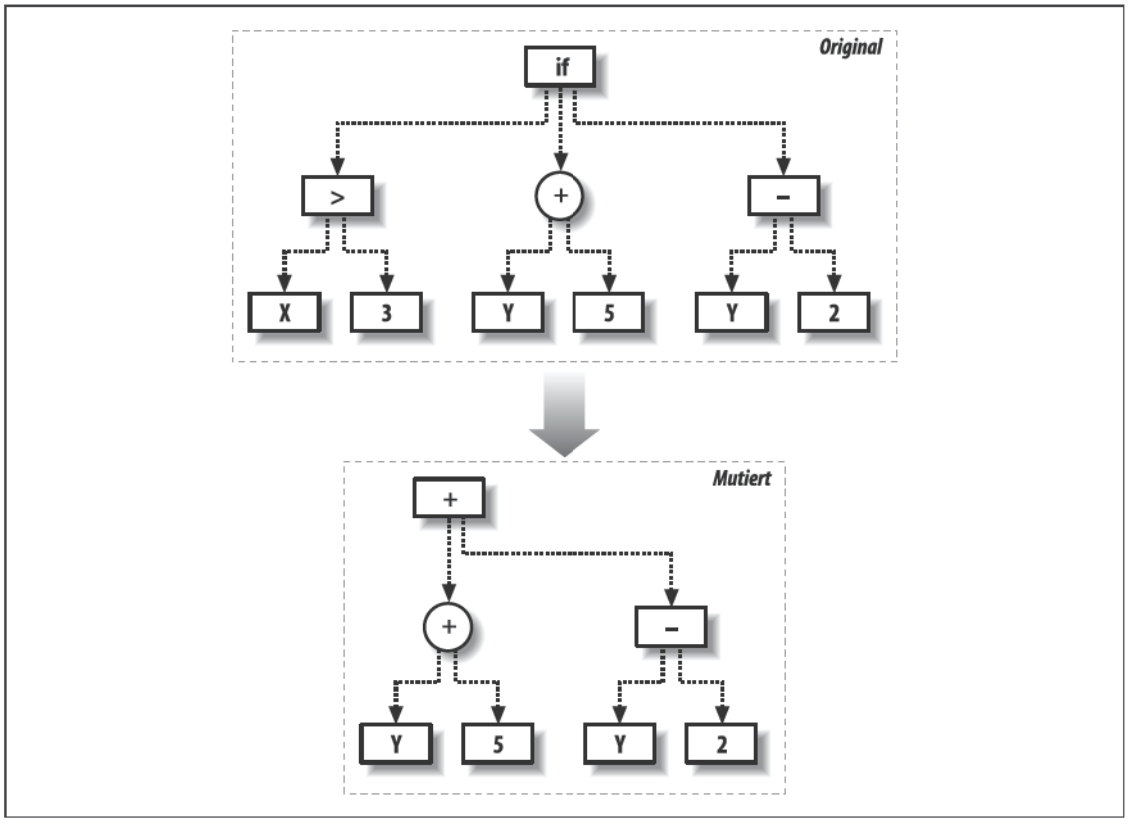

Abbildung 11-3: Mutation durch Ändern der Knotenfunktionen

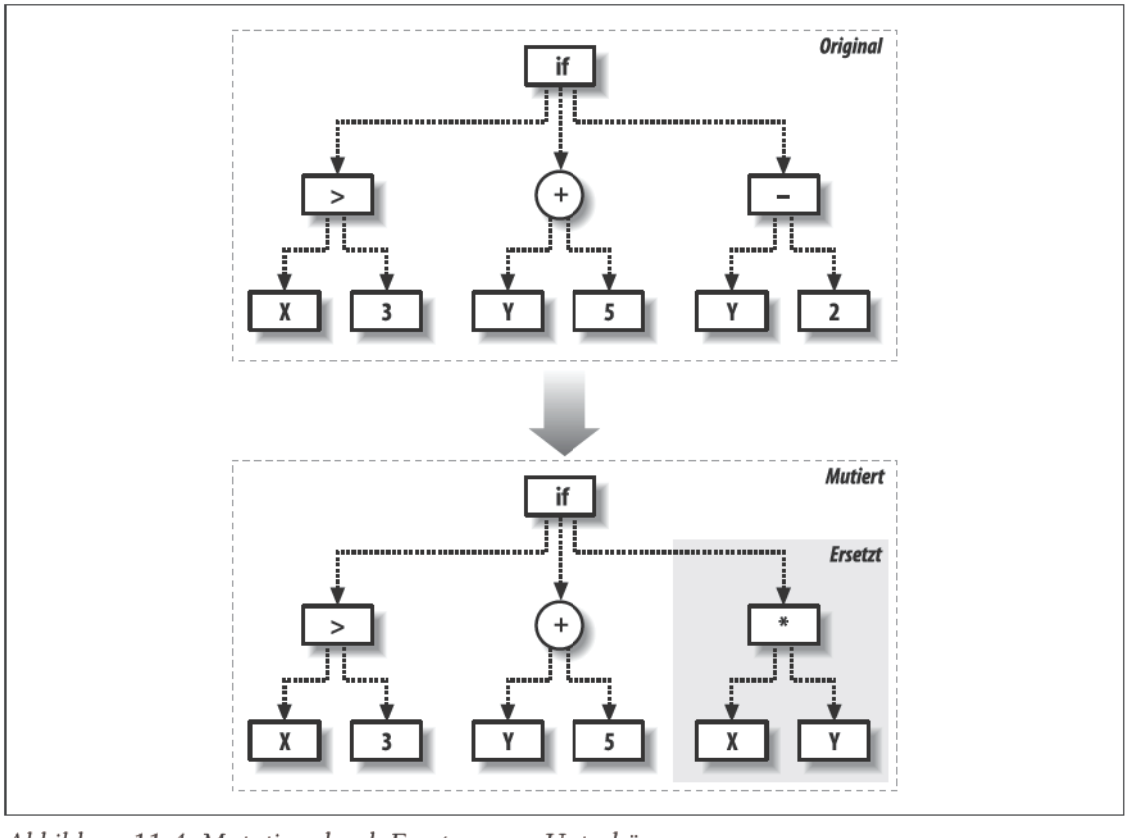

Abbildung 11-4: Mutation durch Ersetzen von Unterbäumen

e-bol.net

Wahrscheinlichkeit vorgeben, dass ein Knoten modifiziert wird. So wird der Baum von seiner Wurzel aus durchlaufen, und bei jedem Knoten wird geprüft, ob eine Zufallszahl kleiner als diese Wahrscheinlichkeit ist. In diesem Fall wird der Knoten auf eine der beschriebenen Arten verändert, ansonsten wird der Test mit einem der Kindknoten fortgeführt.

Um das Ganze nicht zu kompliziert zu machen, führt der hier vorgestellte Code nur die zweite Art von Mutation durch. Erstellen Sie eine neue Funktion namens `mutate`, um diese Operation auszuführen:

```
def mutate(t,pc,probchange=0.1):
  if random( )<probchange:
    return makerandomtree(pc)
  else:
    result=deepcopy(t)
    if isinstance(t,node):
      result.children=[mutate(c,pc,probchange) for c in t.children]
    return result
```

Diese Funktion beginnt am Wurzelknoten und entscheidet, ob er verändert werden sollte. Wenn nicht, ruft sie `mutate` für die Kindknoten des Baums auf. Es kann sein, dass der gesamte Baum verändert wird, genauso kann es aber auch vorkommen, dass der ganze Baum durchlaufen wird, ohne dass sich überhaupt etwas tut.

Lassen Sie `mutate` ein paarmal auf die zufällig generierten Programme los, die Sie zuvor gebaut haben, um zu sehen, wie die Bäume verändert werden:

```
>>> random2.display( )
subtract
 7
 multiply
  isgreater
   p0
   p1
  if
   multiply
    p1
    p1
   p0
   2
>>> muttree=gp.mutate(random2,2)
>>> muttree.display( )
subtract
 7
 multiply
  isgreater
   p0
   p1
  if
   multiply
    p1
    p1
   p0
   p1
```

e-bol.net

Prüfen Sie, ob sich das Ergebnis von `scorefunction` deutlich geändert hat – zum Guten oder zum Schlechten –, nachdem der Baum mutiert wurde:

```
>>> gp.scorefunction(random2,hiddenset)
125489
>>> gp.scorefunction(muttree,hiddenset)
125479
```

Denken Sie daran, dass die Mutationen zufällig sind und daher nicht notwendigerweise die Lösung verbessern. Man kann jedoch nur hoffen, dass eine das Ergebnis verbessern wird. Diese Änderungen werden immer wieder vorgenommen, und nach vielen Generationen wird die beste Lösung schließlich gefunden sein.

Crossover

Die andere Art von Programmänderung ist das Crossover oder Breeding (Züchten). Dazu werden zwei erfolgreiche Programme genommen und zu einem neuen Programm kombiniert, meist, indem man einen Zweig eines Programms durch einen Zweig des anderen Programms ersetzt. Abbildung 11-5 zeigt ein Beispiel dafür.

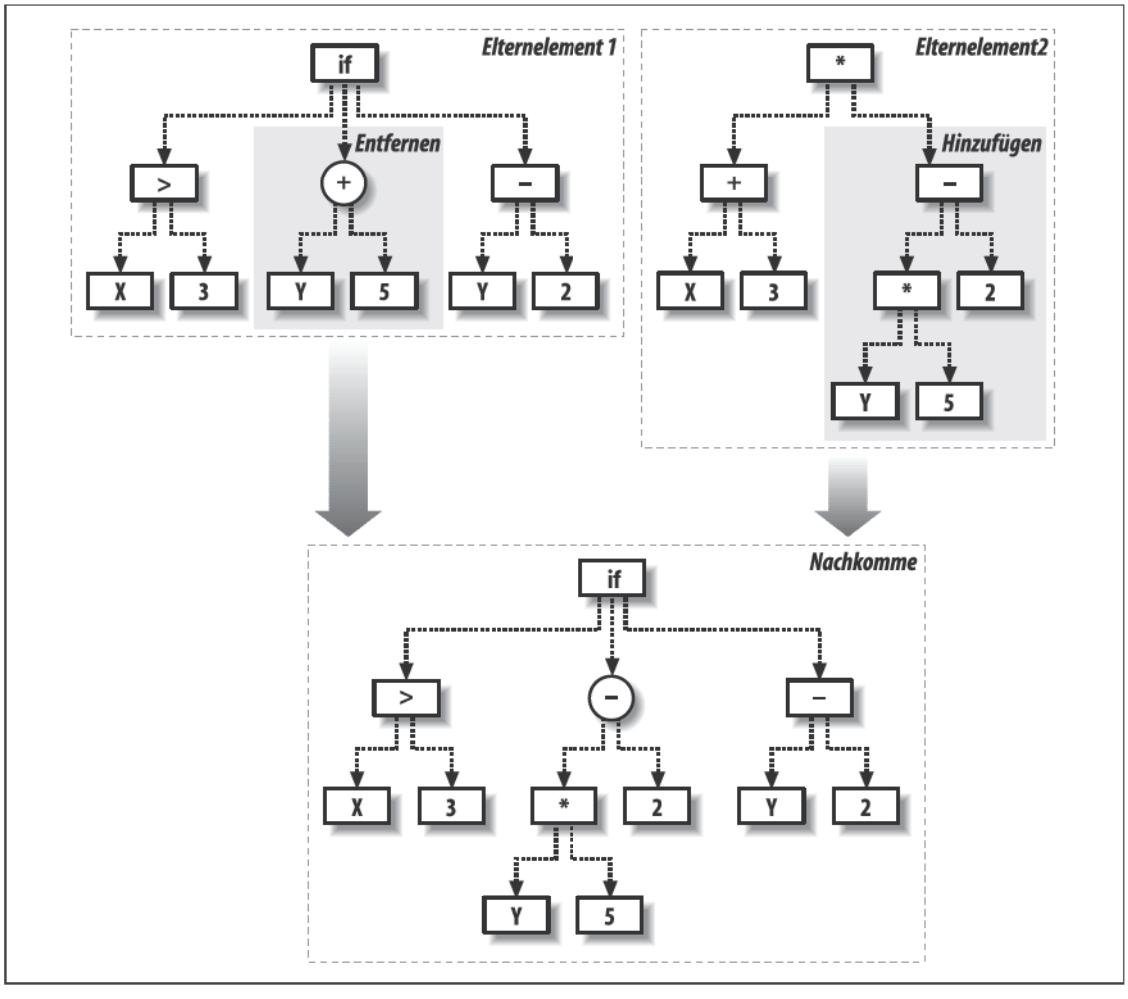

Abbildung 11-5: Crossover-Operation

e-bol.net

Die Funktion zum Durchführen eines Crossover erwartet zwei Bäume als Eingabewerte und durchläuft beide. Wenn ein zufällig gewählter Wert erreicht wird, liefert die Funktion eine Kopie des ersten Baums zurück, wobei einer seiner Zweige durch einen Zweig aus dem zweiten Baum ersetzt wurde. Dadurch, dass beide Bäume gleichzeitig durchlaufen werden, geschieht das Crossover ungefähr auf gleicher Ebene. Fügen Sie die Funktion `crossover` der Datei *gp.py* hinzu:

```
def crossover(t1,t2,probswap=0.7,top=1):
  if random( )<probswap and not top:
    return deepcopy(t2)
  else:
    result=deepcopy(t1)
    if isinstance(t1,node) and isinstance(t2,node):
      result.children=[crossover(c,choice(t2.children),probswap,0)
                       for c in t1.children]
    return result
```

Probieren Sie `crossover` mit ein paar der zufällig generierten Programme aus. Schauen Sie sich die Ergebnisse nach dem Crossover an und prüfen Sie, ob eine Kombination aus den beiden besten Programmen eventuell zu einem noch besseren führt:

```
>>> random1=gp.makerandomtree(2)
>>> random1.display( )
multiply
 subtract
  p0
  8
 isgreater
  p0
  isgreater
   p1
   5
>>> random2=gp.makerandomtree(2)
>>> random2.display( )
if
 8
 p1
 2
>>> cross=gp.crossover(random1,random2)
>>> cross.display( )
multiply
 subtract
  p0
  8
 2
```

Sie werden vermutlich feststellen, dass ein Austauschen von Zweigen die Funktion des Programms komplett verändern kann. Auch werden Sie sehen, dass Programme aus ganz verschiedenen Gründen nahe an der besten Lösung sind und eine Verschmelzung beider zu einem Ergebnis führen kann, das sich von beiden Vorgängern

e-bol.net

massiv unterscheidet. Die Hoffnung ist auch hier wieder, dass ein Crossover die Lösung verbessert und in die nächste Generation übernommen werden kann.

Aufbau der Umgebung

Ausgerüstet mit einer Skala, um den Erfolg zu messen, und zwei Methoden, die die besten Programme modifizieren, sind Sie nun dazu bereit, eine Umgebung einzurichten, in der sich die Programme entwickeln können. Die Schritte sind im Ablaufdiagramm in Abbildung 11-1 zu sehen. Im Prinzip erstellen Sie eine Menge von Zufallsprogrammen und wählen die besten aus, die dann repliziert und modifiziert werden. Dieser Prozess wird wiederholt, bis ein Abbruchkriterium erfüllt ist.

Erstellen Sie eine neue Funktion namens `evolve`, um diese Prozedur durchzuführen:

```
def evolve(pc,popsize,rankfunction,maxgen=500,
           mutationrate=0.1,breedingrate=0.4,pexp=0.7,pnew=0.05):
  # Liefert eine Zufallszahl zurück, wobei kleine Zahlen bevorzugt werden.
  # Je kleiner pexp ist, desto kleinere Zahlen bekommen Sie.
  def selectindex( ):
    return int(log(random( )/log(pexp))

  # Erstellen einer zufälligen Ausgangspopulation.
  population=[makerandomtree(pc) for i in range(popsize)]
  for i in range(maxgen):
    scores=rankfunction(population)
    print scores[0][0]
    if scores[0][0]==0: break

    # Die zwei besten kommen immer durch.
    newpop=[scores[0][1],scores[1][1]]

    # Erstellen der neuen Generation.
    while len(newpop)<popsize:
      if random( )>pnew:
        newpop.append(mutate(
                      crossover(scores[selectindex( )][1],
                                scores[selectindex( )][1],
                               probswap=breedingrate),
                        pc,probchange=mutationrate))
      else:
      # Hinzufügen eines Zufallsknotens, um Abwechslung hineinzubringen.
        newpop.append(makerandomtree(pc))

    population=newpop
  scores[0][1].display( )
  return scores[0][1]
```

Diese Funktion erstellt eine zufällige Ausgangspopulation. Dann läuft sie bis zu `maxgen` Mal, wobei bei jedem Durchlauf `rankfunction` aufgerufen wird, um die Programme vom besten zum schlechtesten zu werten. Das beste Programm wird auto-

e-bol.net

matisch unverändert an die nächste Generation weitergegeben, was manchmal als *Elite* bezeichnet wird. Der Rest der nächsten Generation wird aus zufällig gewählten Programmen aus dem oberen Bereich des Rankings aufgefüllt, wobei diese dann gezüchtet und mutiert werden. Dieser Prozess wiederholt sich, bis entweder ein Programm eine perfekte Bewertung von 0 erreicht oder `maxgen` erreicht wird.

Die Funktion hat diverse Parameter, die genutzt werden können, um die verschiedenen Aspekte der Umgebung zu kontrollieren. Dabei handelt es sich um:

`rankfunction`
: Die Funktion, die die Liste der Programme bewertet.

`mutationrate`
: Die Wahrscheinlichkeit einer Mutation, die an `mutate` übergeben wird.

`breedingrate`
: Die Wahrscheinlichkeit für ein Crossover, die an `crossover` weitergegeben wird.

`popsize`
: Die Größe der Ausgangspopulation.

`probexp`
: Die Rate, mit der die Wahrscheinlichkeit sinkt, dass schlechter bewertete Programme gewählt werden. Ein höherer Wert gestaltet den Auswahlprozess strenger, indem nur Programme mit den besten Bewertungen zur Replikation gewählt werden.

`probnew`
: Die Wahrscheinlichkeit, dass beim Erstellen der neuen Population ein komplett neues zufälliges Programm eingeführt wird. `probexp` und `probnew` werden im noch folgenden Abschnitt »Die Bedeutung der Diversität« besprochen.

Was Sie jetzt noch brauchen, bevor Sie mit der Evolution Ihrer Programme beginnen können, ist eine Möglichkeit, die Programme in Abhängigkeit des Ergebnisses von `scorefunction` zu bewerten. Erstellen Sie in *gp.py* eine neue Funktion namens `getrankfunction`, die eine Bewertungsfunktion für eine gegebene Datenmenge zurückliefert:

```
def getrankfunction(dataset):
  def rankfunction(population):
    scores=[(scorefunction(t,dataset),t) for t in population]
    scores.sort( )
    return scores
  return rankfunction
```

Jetzt können Sie automatisch ein Programm erstellen, das die Formel für Ihre mathematische Datenmenge repräsentiert. Probieren Sie dies in Ihrer Python-Session aus:

e-bol.net

```
>>> reload(gp)
>>> rf=gp.getrankfunction(gp.buildhiddenset( ))
>>> gp.evolve(2,500,rf,mutationrate=0.2,breedingrate=0.1,pexp=0.7,pnew=0.1)
16749
10674
5429
3090
491
151
151
0
add
 multiply
  p0
  add
   2
   p0
 add
  add
   p0
   4
  add
   p1
   add
    p1
    isgreater
     10
     5
```

Die Werte ändern sich langsam, aber sie sollten fallen, bis schließlich 0 erreicht wird. Interessanterweise macht die hier gezeigte Lösung alles richtig, aber sie ist etwas komplizierter als die Funktion, mit der die Daten erzeugt wurden. (Es ist sehr wahrscheinlich, dass die bei Ihnen gefundene Lösung noch komplizierter ist, als sie sein müsste.) Wie auch immer, ein bisschen Algebra zeigt uns, dass diese Funktionen das Gleiche machen – denken Sie daran, dass `p0` für X und `p1` für Y steht. Die erste Zeile ist die Funktion, die von diesem Baum repräsentiert wird:

```
 (X*(2+X))+X+4+Y+Y+(10>5)
= 2*X+X*X+X+4+Y+Y+1
= X**2 + 3*X + 2*Y + 5
```

Dies zeigt eine wichtige Eigenschaft der genetischen Programmierung: Die gefundenen Lösungen mögen korrekt oder zumindest sehr gut sein, aber aufgrund der Art und Weise, wie sie erstellt werden, sind sie häufig weit komplizierter, als sie ein menschlicher Programmierer entwerfen würde. Es wird oft große Bereiche eines Programms geben, die nichts tun oder die eine komplizierte Formel darstellen, die immer den gleichen Wert zurückgibt. Beachten Sie, dass im obigen Beispiel der Knoten `(10>5)` eine etwas schräge Art ist, 1 auszudrücken.

Es ist möglich, die Programme dazu zu zwingen, einfach zu bleiben, aber in vielen Fällen wird es dadurch deutlich schwieriger werden, eine gute Lösung zu finden.

e-bol.net

Man geht das besser an, indem man den Programmen erlaubt, eine gute Lösung zu entwickeln und danach die unnötigen Bereiche des Baums zu vereinfachen oder zu entfernen. Sie können das manuell erledigen, in manchen Fällen können Sie das aber auch automatisch mit einem Pruning-Algorithmus erreichen.

Die Bedeutung der Diversität

Ein Teil der Funktion `evolve` bewertet und ordnet die Programme vom besten bis zum schlechtesten, daher ist es verführerisch, nur die zwei oder drei Programme an der Spitze zu nehmen und diese zu replizieren und zu modifizieren, um daraus die neue Population zu erstellen. Warum sollten Sie schließlich irgendetwas weiterleben lassen, das nicht das Beste ist?

Das Problem liegt darin, dass man beim Wählen von nur wenigen der besten Lösungen die Population schnell außerordentlich homogen werden lässt (man kann es auch Inzucht nennen) und dadurch Lösungen versammelt, die zwar alle ziemlich gut sind, sich aber nicht stark ändern werden, weil die Crossover-Operationen zwischen ihnen keine großen Abwandlungen bewirken. Dieses Problem wird als Erreichen eines *lokalen Maximums* bezeichnet, eines Zustands, der zwar gut, aber nicht gut genug ist. Dabei sorgen kleine Veränderungen für keine weitere Verbesserung des Ergebnisses.

Es zeigt sich, dass eine Kombination aus den besten Lösungen und einer großen Zahl von halbwegs guten Lösungen im Allgemeinen zu besseren Ergebnissen führt. Aus diesem Grund hat die Funktion `evolve` ein paar zusätzliche Parameter, mit denen Sie am Umfang der Diversität im Selektionsprozess drehen können. Indem Sie den Wert `probexp` reduzieren, ermöglichen Sie es schwächeren Lösungen, auch in die nächste Generation zu gelangen, wodurch Sie den Prozess vom »Überleben der Stärksten« in »Überleben der Stärksten und Glücklichsten« umwandeln. Erhöhen Sie den Wert `probnew`, erlauben Sie vollständig neuen Programmen, ab und zu dem Mix hinzugefügt zu werden. Beide Werte erhöhen die Diversität im Evolutionsprozess, zerstören aber nicht die gesamte Prozedur, da die wirklich schlechten Programme zum Schluss immer eliminiert werden.

Ein einfaches Spiel

Ein interessanteres Problem für die genetische Programmierung ist das Erstellen einer KI für ein Spiel. Sie können die Programme dazu bringen, sich zu entwickeln, indem Sie sie gegeneinander und gegen echte Personen antreten lassen, und denjenigen Programmen eine höhere Chance geben, es in die nächste Generation zu schaffen, die meistens gewinnen. In diesem Abschnitt werden Sie einen Simulator für ein sehr einfaches Spiel namens Grid War schreiben, das in Abbildung 11-6 zu sehen ist.

e-bol.net

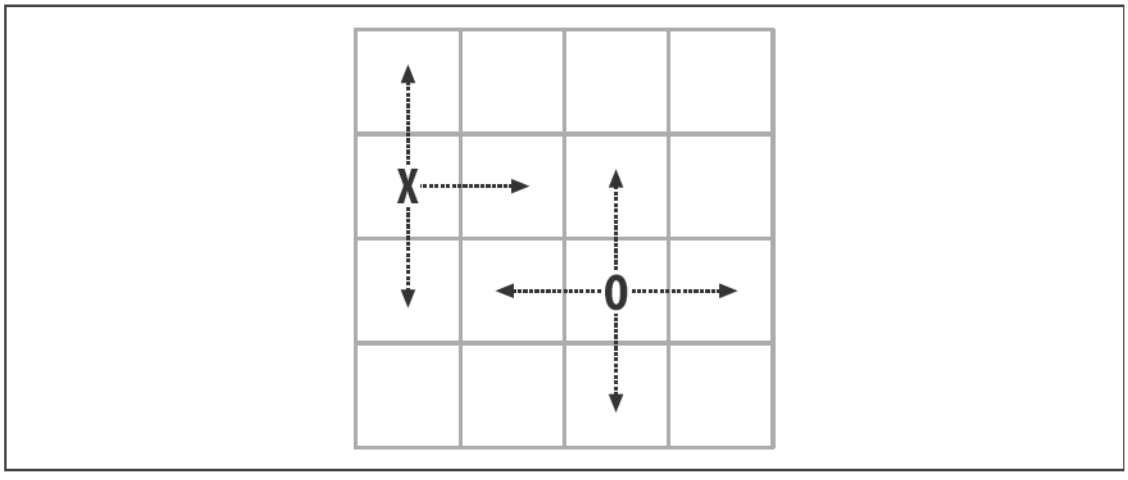

Abbildung 11-6: Beispiel für Grid War

Das Spiel hat zwei Spieler, die abwechselnd ihren Spielstein auf einem kleinen Raster bewegen. Jeder Spieler kann in eine der vier Richtungen ziehen, wobei das Brett begrenzt ist und man nicht über den Rand hinausziehen kann. Ziel des Spiels ist, den anderen Spieler zu fangen, indem man seinen Stein mit seinem Zug auf das Quadrat zieht, auf dem der Gegner steht. Die einzige zusätzliche Regel ist, dass man automatisch verliert, wenn man versucht, zweimal hintereinander in die gleiche Richtung zu ziehen. Dieses Spiel ist sehr einfach, aber da nur zwei Spieler gegeneinander antreten, können Sie die Wettbewerbsaspekte der Evolution besser untersuchen.

Zuerst erstellt man eine Funktion, die zwei Spieler »erzeugt« und ein Spiel zwischen ihnen simuliert. Die Funktion übergibt die Position des Spielers und des Gegners zusammen mit dem letzten Zug an jedes Programm und nimmt den Rückgabewert als Zug entgegen.

Der Zug sollte eine Zahl zwischen 0 und 3 sein, womit eine der vier möglichen Richtungen angegeben wird, aber da es sich um Zufallsprogramme handelt, die beliebige Integer-Werte zurückgeben, muss die Funktion auch mit Werten außerhalb dieses Bereichs umgehen können. Dazu verwendet es beim Ergebnis *modulo 4*. Zufallsprogramme laufen auch Gefahr, beispielsweise einen Spieler zu erzeugen, der sich im Kreis bewegt, daher ist die Anzahl an Zügen auf 50 beschränkt, bevor ein Remis festgestellt wird.

Fügen Sie `gridgame` der Datei *gp.py* hinzu:

```
def gridgame(p):
  # Brettgröße
  max=(3,3)

  # Den letzten Zug für jeden Spieler merken.
  lastmove=[-1,-1]

  # Die Position des Spielers festlegen.
```

e-bol.net

```
    location=[[randint(0,max[0]),randint(0,max[1])]]

    # Den zweiten Spieler mit gewissem Abstand vom ersten platzieren.
    location.append([(location[0][0]+2)%4,(location[0][1]+2)%4])
    # Die höchsten 50 Züge, bis Gleichstand festgestellt wird.
    for o in range(50):

      # Für jeden Spieler.
      for i in range(2):
        locs=location[i][:]+location[1-i][:]
        locs.append(lastmove[i])
        move=p[i].evaluate(locs)%4

        # Sie verlieren, wenn Sie zweimal in die gleiche Richtung gehen.
        if lastmove[i]==move: return 1-i
        lastmove[i]=move
        if move==0:
          location[i][0]-=1
          # Brettgrenzen
          if location[i][0]<0: location[i][0]=0
        if move==1:
          location[i][0]+=1
          if location[i][0]>max[0]: location[i][0]=max[0]
        if move==2:
          location[i][1]-=1
          if location[i][1]<0: location[i][1]=0
        if move==3:
          location[i][1]+=1
          if location[i][1]>max[1]: location[i][1]=max[1]

        # Wenn Sie den anderen Spieler gefangen haben, gewinnen Sie.
        if location[i]==location[1-i]: return i
    return -1
```

Das Programm liefert eine 0 zurück, wenn Spieler 1 gewinnt, 1, wenn Spieler 2 gewinnt, und –1 bei einem Remis. Sie können versuchen, ein paar Zufallsprogramme zu erzeugen, und sie gegeneinander antreten lassen:

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> p1=gp.makerandomtree(5)
>>> p2=gp.makerandomtree(5)
>>> gp.gridgame([p1,p2])
1
```

Diese Programme sind vollständig unentwickelt, daher werden sie sehr wahrscheinlich verlieren, weil sie zweimal nacheinander in die gleiche Richtung ziehen. Idealerweise wird ein entwickeltes Programm lernen, das nicht zu tun.

Ein Turnier »jeder gegen jeden«

Um im Rahmen der kollektiven Intelligenz zu bleiben, würden Sie sicherlich die Programme ihre Fähigkeiten austesten lassen wollen, indem Sie sie gegen echte Per-

e-bol.net

sonen spielen lassen und damit ihre Entwicklung fördern. Das wäre eine tolle Möglichkeit, das Verhalten Tausender von Personen zu erfassen und damit intelligentere Programme zu entwickeln. Aber bei solch einer großen Population und vielen Generationen führt dies schnell zu Zehntausenden von Spielen, bei denen meistens gegen sehr schlechte Gegner gespielt würde. Das ist für unsere Zwecke nicht umsetzbar, daher können Sie die Programme sich erst selbst entwickeln lassen, indem Sie sie in einem Turnier gegeneinander antreten lassen.

Die Funktion `tournament` erwartet eine Liste von Spielern als Eingabe und lässt jeden gegen jeden antreten, wobei sie aufzeichnet, wie oft jedes Programm ein Spiel verliert. Programme erhalten zwei Punkte, wenn sie verlieren, und einen bei Gleichstand. Fügen Sie `tournament` der Datei *gp.py* hinzu:

```
def tournament(pl):
  # Verlorene Spiele zählen.
  losses=[0 for p in pl]

  # Jeder Spieler spielt gegen jeden anderen.
  for i in range(len(pl)):
    for j in range(len(pl)):
      if i==j: continue

      # Wer ist der Gewinner?
      winner=gridgame([pl[i],pl[j]])

      # Zwei Punkte für eine Niederlage, einen für einen Gleichstand.
      if winner==0:
        losses[j]+=2
      elif winner==1:
        losses[i]+=2
      elif winner==-1:
        losses[i]+=1
        losses[i]+=1
        pass

  # Sortieren und Zurückgeben der Ergebnisse.
  z=zip(losses,pl)
  z.sort()
  return z
```

Am Ende der Funktion werden die Ergebnisse sortiert und zurückgegeben, wobei die Programme mit den wenigsten verlorenen Spielen an der Spitze stehen. Das ist ein Rückgabewert, wie er von `evolve` benötigt wird, um Programme zu evaluieren. Damit kann `tournament` als Argument von `evolve` genutzt werden, und Sie sind bereits dazu in der Lage, ein Programm zum Spielen des Spiels entwickeln zu lassen. Probieren Sie dies in Ihrer Python-Session aus (es kann einige Zeit dauern):

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> winner=gp.evolve(5,100,gp.tournament,maxgen=50)
```

e-bol.net

Während sich die Programme entwickeln, stellen Sie vielleicht fest, dass die Anzahl der verlorenen Spiele nicht immer sinkt, wie dies bei der mathematischen Funktion der Fall war. Überlegen Sie mal, warum das so ist – schließlich schafft es der beste Spieler immer in die nächste Generation, oder? Wie sich herausstellt, kann das beste Programm in einer Generation in der nächsten völlig abschmieren, weil diese neue Generation aus vollständig neu entwickelten Programmen besteht.

Spielen gegen reale Gegner

Nachdem Sie ein Programm entwickelt haben, das sich gegen seine Computergegner gut wehren kann, ist es an der Zeit, es gegen sich selbst spielen zu lassen. Dazu können Sie eine weitere Klasse erstellen, die auch eine Evaluierungsmethode besitzt, das Brett dem Benutzer präsentiert und ihn fragt, welchen Zug er machen möchte. Fügen Sie die Klasse `humanplayer` *gp.py* hinzu:

```
class humanplayer:
  def evaluate(self,board):

    # Eigene Position und Position des anderen Spielers erhalten.
    me=tuple(board[0:2])
    others=[tuple(board[x:x+2]) for x in range(2,len(board)-1,2)]

    # Anzeigen des Bretts.
    for i in range(4):
      for j in range(4):
        if (i,j)==me:
          print 'O',
        elif (i,j) in others:
          print 'X',
        else:
          print '.',
      print

    # Züge zur Referenz anzeigen.
    print 'Ihr letzter Zug war %d' % board[len(board)-1]
    print ' 0'
    print '2 3'
    print ' 1'
    print 'Zug eingeben: ',

    # Zurückgeben, was der Benutzer eingibt.
    move=int(raw_input())
    return move
```

In Ihrer Python-Session können Sie nun spielen:

```
>>> reload(gp)
<module 'gp' from 'gp.py'>
>>> gp.gridgame([winner,gp.humanplayer()])
. O . .
. . . .
```

e-bol.net

```
. . . .
. . . X
Your last move was -1
 0
2 3
 1
Enter move:
```

Abhängig davon, wie gut sich Ihr Programm entwickelt hat, mögen Sie es einfach oder schwierig finden, gegen es zu gewinnen. Ihr Programm wird mit Sicherheit gelernt haben, dass es den gleichen Zug nicht zweimal hintereinander ausführen darf, da es damit sofort verliert. Aber das Beherrschen anderer Strategien steigt mit jedem Durchlauf von `evolve`.

Weitere Möglichkeiten

Dieses Kapitel ist nur eine Einführung in die genetische Programmierung, bei der es sich um einen riesigen und schnell wachsenden Bereich handelt. Sie haben sie bisher genutzt, um einfache Probleme anzugehen, bei denen Programme in Minuten statt in Tagen entstehen, aber die Prinzipien lassen sich auf deutlich komplexere Probleme ausdehnen. Die Anzahl an Programmen in den Populationen ist hier bisher sehr klein gewesen – im Vergleich zu den Mengen in komplexeren Problemen –, eine Population mit Tausenden oder Zehntausenden von Programmen kommt deutlich häufiger vor. Sie sind jetzt dazu in der Lage, schwierigere Probleme anzugehen und größere Populationen zu verwenden, aber es kann sein, dass Sie Stunden oder Tage warten müssen, während die Programme laufen.

Der folgende Abschnitt präsentiert ein paar Möglichkeiten, in denen das einfache genetische Programmiermodell auf andere Anwendungen ausgeweitet werden kann.

Mehr numerische Funktionen

Wir haben bisher nur eine sehr kleine Menge an Funktionen genutzt, um die Programme zu erstellen. Damit beschränkt man sich auf einfache Programme – bei komplizierteren Problemen ist es notwendig, die Anzahl der zum Aufbau eines Baums verfügbaren Funktionen zu erhöhen. Hier ein paar mögliche Funktionen, die man hinzufügen kann:

- Trigonometrische Funktionen wie Sinus, Kosinus und Tangens.
- Andere mathematische Funktionen wie Potenzen, Wurzeln und den Absolutbetrag.
- Statistische Verteilungen wie die Gauß-Verteilung.
- Abstandsmetriken, wie der euklidische und der Tanimoto-Abstand.

e-bol.net

- Eine Funktion mit drei Parametern, die 1 zurückgibt, wenn der erste Parameter zwischen dem zweiten und dritten liegt.
- Eine Funktion mit drei Parametern, die 1 zurückgibt, wenn die Differenz zwischen den ersten beiden Parametern kleiner als der dritte ist.

Diese Funktionen können so kompliziert werden, wie Sie möchten, und sie werden häufig auf die spezifischen Probleme zugeschnitten. Trigonometrische Funktionen können notwendig sein, wenn Sie zum Beispiel mit Signalverarbeitung zu tun haben, sind aber bei Spielen wie dem hier noch zu erstellenden nicht so nützlich.

Speicher

Die Programme in diesem Kapitel reagieren fast sofort – sie liefern ein Ergebnis zurück, das nur von den Eingaben abhängt. Das ist der richtige Ansatz, wenn Sie mathematische Funktionen lösen wollen, ermöglicht den Programmen aber nicht, an einer längerfristigen Strategie zu arbeiten. Das Fangen-Spiel übergibt den Programmen den letzten Zug – vor allem damit die Programme lernen können, dass sie den gleichen Zug nicht zweimal hintereinander ausführen sollen – aber das ist nur die Ausgabe des Programms und nichts, was sie selbst festgelegt haben.

Wenn ein Programm eine längerfristige Strategie entwickeln können soll, braucht es eine Möglichkeit, Informationen abzuspeichern, die in der nächsten Runde gebraucht werden. Eine einfache Möglichkeit dazu ist, neue Arten von Knoten zu erstellen, die Werte in festgelegten Fächern ablegen und zurückgeben können. Ein *Store*-Knoten hat ein einzelnes Kindelement und einen Index für einen Speicherplatz – er erhält die Ergebnisse von seinem Kindknoten, legt ihn im Speicherplatz ab und übergibt ihn an seinen Elternknoten. Ein *Recall*-Knoten hat keine Kinder und liefert einfach den Wert im entsprechenden Fach zurück. Wenn der Wurzelknoten ein Store-Knoten ist, ist das Endergebnis für alle Teile des Baums verfügbar, die einen entsprechenden Recall-Knoten enthalten.

Neben solchen individuellen Speicherfächern ist es auch möglich, gemeinsam genutzte Speicherbereiche (Shared Memory) einzurichten, die von allen Programmen gelesen und geschrieben werden können. Damit bietet man Potenzial für mehr Kooperation und Wettbewerb.

Unterschiedliche Datentypen

Das Framework, das in diesem Kapitel beschrieben wurde, ist für Programme gedacht, die Integer-Werte erwarten und auch Integer-Werte als Ergebnisse zurückliefern. Es lässt sich leicht anpassen, um auch mit Gleitkommazahlen zu arbeiten, da die Operationen die gleichen sind. Dazu passen Sie nur `makerandomtree` an, um die konstanten Knoten mit einem zufälligen Gleitkommawert zu initialisieren statt mit einem zufälligen Integer-Wert.

e-bol.net

Das Erstellen von Programmen, die andere Arten von Daten verarbeiten können, erfordert umfangreichere Anpassungen, vor allem bei den Funktionen der Knoten. Das einfache Framework kann so angepasst werden, dass es mit verschiedenen Typen umgeht:

Strings
: Dazu gehören Operationen wie Verbinden, Auftrennen, Indexieren und Substrings.

Listen
: Dazu passen ähnliche Operationen wie bei den Strings.

Dictionaries
: Hier passen Operationen wie Ersetzen und Hinzufügen.

Objekte
: Jedes eigene Objekt kann als Eingabe für einen Baum dienen, wobei die Funktionen an den Knoten Methodenaufrufe des Objekts sind.

Ein wichtiger Punkt, der sich aus diesen Beispielen ergibt, ist, dass die Knoten im Baum in vielen Fällen mehr als einen Typ von Rückgabewerten verarbeiten können müssen. Eine Operation für Substrings erfordert zum Beispiel einen String und zwei Integer-Werte, was bedeutet, dass eines der Kindelemente einen String und die anderen beiden Integer-Werte liefern müssen.

Ein einfacher Ansatz dafür wäre es, bei dem zufälligen Erzeugen, Mutieren und Züchten von Bäumen diejenigen auszusondern, bei denen die Datentypen nicht passen. Aber das wäre eine echte Verschwendung von Rechenleistung, und Sie haben schon gesehen, wie Sie die Bäume beim »Wachsen« einschränken können – jede Funktion in den Integer-Bäumen weiß, wie viele Kindknoten sie benötigt, was sich leicht auf den Typ der Kindelemente und ihre Rückgabewerte erweitern lässt. So könnten Sie zum Beispiel die Klasse `fwrapper` wie folgt umbauen, wobei `params` eine Liste mit Strings enthält, die die Datentypen für jeden Parameter angeben:

```
class fwrapper:
  def __init__(self,function,params,name):
    self.function=function
    self.childcount=param
    self.name=name
```

Sie werden dann wahrscheinlich auch `flist` als Dictionary mit Rückgabewerten füllen, zum Beispiel:

```
flist={'str':[substringw,concatw],'int':[indexw,addw,subw]}
```

Dann könnten Sie den Anfang von `makerandomtree` so ändern:

```
def makerandomtree(pc,datatype,maxdepth=4,fpr=0.5,ppr=0.5):
  if random( )<fpr and maxdepth>0:
    f=choice(flist[datatype])
    # makerandomtree mit allen Parametertypen für f aufrufen.
```

e-bol.net

```
        children=[makerandomtree(pc,type,maxdepth-1,fpr,ppr)
                  for type in f.params]
        return node(f,children)
    usw.
```

Die Funktion `crossover` müsste ebenfalls angepasst werden, um sicherzustellen, dass ausgetauschte Knoten den gleichen Rückgabetyp haben.

Idealerweise hat Ihnen dieser Abschnitt ein paar Anregungen dazu gegeben, wie die genetische Programmierung aus dem einfachen, hier beschriebenen Modell heraus erweitert werden kann, und hat Sie dazu inspiriert, das Modell zu verbessern und automatisch generierte Programme für komplexere Probleme erzeugen zu lassen. Auch wenn es sehr lange dauern kann, bis die Generierung fertig ist – wenn Sie einmal ein gutes Programm gefunden haben, können Sie es wieder und wieder nutzen.

Übungen

1. *Mehr Funktionstypen.* Wir haben mit einer sehr kurzen Liste mit Funktionen begonnen. Welche anderen Funktionen könnte man noch nutzen? Implementieren Sie einen Knoten für die euklidische Distanz mit vier Parametern.
2. *Mutation zum Ersetzen.* Implementieren Sie eine Mutationsprozedur, die zufällig einen Knoten auswählt und ihn ändert. Stellen Sie sicher, dass sie mit Funktions-, Konstanten- und Parameterknoten umgehen kann. Wie wird die Evolution beeinflusst, wenn diese Funktion statt des Austauschs eines Zweigs verwendet wird?
3. *Zufälliges Crossover.* Die aktuelle Crossover-Funktion wählt Zweige aus zwei Bäumen auf der gleichen Ebene. Schreiben Sie eine andere Funktion, die zwei beliebige Zweige miteinander vertauscht. Wie wirkt sich das auf die Evolution aus?
4. *Beenden der Evolution.* Fügen Sie `evolve` ein zusätzliches Kriterium hinzu, das den Prozess abbricht und das beste Ergebnis zurückliefert, wenn sich dieses seit x Generationen nicht verbessert hat.
5. *Verborgene Funktionen.* Versuchen Sie, die Programme andere mathematische Funktionen für die Funktionen finden zu lassen. Was für Arten von Funktionen können leicht gefunden werden, und bei welchen ist es schwieriger?
6. *Grid-War-Spieler.* Versuchen Sie, Ihr eigenes Baumprogramm per Hand zu erstellen, das sich bei Grid War gut schlägt. Wenn Sie das leicht finden, versuchen Sie, ein anderes, komplett unterschiedliches Programm zu schreiben. Anstatt eine vollständig zufällige erste Population zu erstellen, lassen Sie sie nur *fast* zufällig entstehen, indem Ihre selbst geschriebenen Programme enthalten sind. Wie machen sie sich im Vergleich zu den Zufallsprogrammen, und werden sie durch die Evolution noch verbessert?

e-bol.net

7. *Tic-Tac-Toe*. Schreiben Sie einen Tic-Tac-Toe-Simulator für Ihre Programme, damit diese spielen können. Setzen Sie einen Wettbewerb auf, der so wie bei Grid War funktioniert. Wie schlagen sich die Programme? Werden sie jemals lernen, perfekt zu spielen?
8. *Knoten mit Datentypen*. In diesem Kapitel wurden ein paar Ideen dazu vorgestellt, Knoten mit unterschiedlichen Datentypen zu implementieren. Setzen Sie das um und versuchen Sie, ein Programm entwickeln zu lassen, das lernt, das zweite, dritte, sechste und siebte Zeichen eines Strings zurückzugeben (»genetic« wird also zu »enic«).

e-bol.net

KAPITEL 12

Zusammenfassung der Algorithmen

In diesem Buch wurden eine Reihe unterschiedlicher Algorithmen vorgestellt, und wenn Sie die Kapitel durchgearbeitet haben, steht Ihnen nun Python-Code zur Verfügung, mit dem die meisten von ihnen umgesetzt werden können. Die bisherigen Kapitel waren dergestalt aufgebaut, dass ein Beispielproblem durch Algorithmen gelöst wurde und im Laufe des Kapitels verschiedene Varianten vorgestellt wurden. Dieses Kapitel soll eine Referenz für die behandelten Algorithmen sein – wenn Sie also Datamining oder maschinelles Lernen für eine neue Datenmenge umsetzen wollen, können Sie sich die Algorithmen hier anschauen, sich überlegen, ob ein passender darunter ist, und den schon geschriebenen Code nutzen, um Ihre Daten zu analysieren.

Um es Ihnen zu ersparen, im ganzen Buch nach den Details eines Algorithmus suchen zu müssen, werde ich für jeden eine Beschreibung, einen groben Überblick über seine Funktionsweise, die Art der Daten, auf die er angewandt werden kann, und die Verwendung des Codes, den Sie dafür geschrieben haben, liefern. Ich werde auch einige der Stärken und Schwächen jedes Algorithmus darlegen (oder wie Sie ihn Ihrem Chef schmackhaft machen können, wenn Sie das wollen). In manchen Fällen werde ich Beispiele nutzen, um die Eigenschaften des Algorithmus besser beschreiben zu können. Diese Beispiele sind stark vereinfacht – die meisten sind so einfach, dass Sie sie durch bloßes Betrachten lösen können –, aber sie sind zu Demonstrationszwecken sehr nützlich.

Methoden zum überwachten Lernen, die eine Klassifikation oder einen Wert in Abhängigkeit von Trainingsdaten vorschlagen, werden als Erstes behandelt.

Bayes-Klassifizierer

Bayes-Klassifizierer wurden in Kapitel 6 behandelt. Dort haben Sie gesehen, wie Sie ein Dokumenten-Klassifikationssystem erstellen können – beispielsweise zum Fil-

e-bol.net

tern von Spam oder zum Aufteilen einer Reihe von Dokumenten basierend auf einer mehrdeutigen Schlüsselwortsuche.

Auch wenn es in allen Beispielen um Dokumente ging, funktioniert der in Kapitel 6 beschriebene Bayes-Klassifizierer mit allen Daten, die in *Merkmals*-Listen umgewandelt werden können. Ein Merkmal ist einfach etwas, das für ein gegebenes Element entweder vorhanden ist oder fehlt. Bei Dokumenten handelt es sich bei den Merkmalen um die Wörter des Dokuments, aber es könnte sich auch um Eigenschaften eines unbekannten Objekts handeln, um die Symptome einer Krankheit oder etwas anderes, solange man bestimmen kann, ob es da ist oder nicht.

Training

Wie bei allen überwachten Methoden wird ein Bayes-Klassifizierer durch Beispiele trainiert. Jedes Beispiel ist eine Liste mit den Merkmalen eines Elements und seiner Klassifikation. Stellen Sie sich vor, Sie würden einen Klassifizierer darauf trainieren zu erkennen, ob ein Dokument mit dem Wort »python« von der Programmiersprache oder der Schlange handelt. Eine Beispiel-Datenmenge könnte so aussehen wie das in Tabelle 12-1.

Tabelle 12-1: Merkmale und Klassifikationen für eine Reihe von Dokumenten

Merkmale	Klassifikation
Pythons are constrictors that feed on birds and mammals	Schlange
Python was originally developed as a scripting language	Sprache
A 49-ft.-long python was found in Indonesia	Schlange
Python has a dynamic type system	Sprache
Python with vivid scales	Schlange
Open source project	Sprache

Der Klassifizierer merkt sich alle Merkmale, die ihm bisher vorgesetzt wurden, zusammen mit Wahrscheinlichkeitswerten, mit denen die Merkmale mit einer bestimmten Klassifikation verbunden sind. Der Klassifizierer wird trainiert, indem er nacheinander Beispiele vorgesetzt bekommt. Nach jedem Beispiel passt der Klassifizierer die Wahrscheinlichkeiten für die Merkmale und die Klasse an und erstellt damit die Wahrscheinlichkeit, dass ein Dokument einer bestimmten Kategorie ein gegebenes Wort enthält. Nimmt man zum Beispiel die Dokumente aus Tabelle 12-1, erhält man schließlich eine Reihe von Wahrscheinlichkeiten, die in Tabelle 12-2 zu sehen sind.

e-bol.net

Tabelle 12-2: Wahrscheinlichkeiten für Wörter für eine bestimmte Kategorie

Merkmal	Sprache	Schlange
dynamic	0,6	0,1
constrictor	0,0	0,6
long	0,1	0,2
source	0,3	0,1
and	0,95	0,95

Diese Tabelle zeigt, dass die Merkmale nach dem Training enger mit den verschiedenen Kategorien verbunden werden. Das Wort »constrictor« hat eine höhere Wahrscheinlichkeit für Schlangen und das Wort »dynamic« eine höhere für die Programmiersprache. Merkmale, die nicht eindeutig sind, wie zum Beispiel das Wort »and«, haben gleiche Wahrscheinlichkeiten für beide Kategorien (das Wort »and« taucht in so gut wie jedem Dokument auf, unabhängig von seiner Kategorie). Ein trainierter Klassifizierer ist nicht mehr als eine Liste von Merkmalen zusammen mit ihren zugehörigen Wahrscheinlichkeiten. Anders als manch andere Klassifikationsmethoden gibt es keine Notwendigkeit, die ursprünglichen Daten nach deren Nutzung für das Training abzuspeichern.

Klassifizieren

Nach dem Training kann ein Bayes-Klassifizierer genutzt werden, um automatisch neue Elemente zu klassifizieren. Stellen Sie sich vor, Sie hätten ein neues Dokument mit den Merkmalen »long«, »dynamic« und »source«. Tabelle 12-2 zeigt einen Wahrscheinlichkeitswert für jedes dieser Merkmale, aber er bezieht sich nur auf die einzelnen Wörter. Wenn alle Wörter eine größere Wahrscheinlichkeit in einer Kategorie haben, ist die Antwort klar. Aber in diesem Fall hat »dynamic« eine größere Wahrscheinlichkeit für die Kategorie Sprache, während »long« eher in die Schlangen-Kategorie einzuordnen ist. Um ein Dokument nun tatsächlich zu klassifizieren, brauchen Sie eine Möglichkeit, die Merkmalswahrscheinlichkeiten in einer einzelnen Wahrscheinlichkeit für das gesamte Element zu kombinieren.

Eine in Kapitel 6 dafür beschriebene Vorgehensweise ist der naive Bayes-Klassifizierer. Er verbindet die Wahrscheinlichkeiten mit folgender Formel:

$$Pr(Kategorie \mid Dokument) = Pr(Dokument \mid Kategorie) * Pr(Kategorie)$$

wobei gilt:

$$Pr\ (Dokument \mid Kategorie) = Pr(Wort1 \mid Kategorie) * Pr(Wort2 \mid Kategorie) * \ldots$$

Die Werte für *Pr(Wort | Kategorie)* stammen aus der Tabelle: So gilt zum Beispiel *Pr (dynamic | Sprache)* = 0,6. Der Wert *Pr(Kategorie)* ist die Gesamthäufigkeit der Kategorie. Da »Sprache« in der Hälfte der Fälle auftritt, hat *Pr(Sprache)* den Wert

e-bol.net

0,5. Die Kategorie, die einen höheren Wert *Pr(Kategorie | Dokument)* hat, ist die vorhergesagte Kategorie.

Verwenden Ihres Codes

Um den in Kapitel 6 aufgebauten Bayes-Klassifizierer für eine beliebige Datenmenge nutzen zu können, brauchen Sie nur eine Funktion zur Merkmalsextraktion, die die zum Trainieren oder Klassifizieren genutzten Daten in eine Liste mit Merkmalen umwandelt. Das Kapitel hat mit Dokumenten gearbeitet, daher teilte die Funktion einen String in Wörter auf, aber Sie können eine beliebige Funktion nutzen, die ein Objekt erwartet und eine Liste zurückgibt:

```
>>> docclass.getwords('python is a dynamic language')
{'python': 1, 'dynamic': 1, 'language': 1}
```

Diese Funktion kann genutzt werden, um einen neuen Klassifizierer zu erstellen, der dann mit Strings trainiert wird:

```
>>> cl=docclass.naivebayes(docclass.getwords)
>>> cl.setdb('test.db')
>>> cl.train('pythons are constrictors','Schlange')
>>> cl.train('python has dynamic types','Sprache')
>>> cl.train('python was developed as a scripting language','Sprache')
```

sowie zur Klassifikation:

```
>>> cl.classify('dynamic programming')
u'Sprache'
>>> cl.classify('boa constrictors')
u'Schlange'
```

Es gibt keine Obergrenze bei der Anzahl der verschiedenen Kategorien, aber damit der Klassifizierer gut funktioniert, brauchen Sie eine ganze Reihe von Beispielen für jede Kategorie.

Stärken und Schwächen

Der vielleicht größte Vorteil des naiven Bayes-Klassifizierers gegenüber anderen Methoden ist die Geschwindigkeit, mit der er bei großen Datenmengen trainiert und abgefragt werden kann. Selbst bei einem riesigen Trainingsset gibt es meist nur recht wenige Merkmale für jedes Element, und das Trainieren und Klassifizieren von Elementen beinhaltet lediglich Änderungen an den Wahrscheinlichkeiten dieser Merkmale.

Das gilt insbesondere dann, wenn das Training nach und nach stattfindet – jeder neue Trainingshappen kann genutzt werden, um die Wahrscheinlichkeiten anzupassen, ohne auf alte Trainingsdaten zurückgreifen zu müssen. (Sie werden feststellen, dass Sie mit dem Code immer jeweils ein Element zum Trainieren nutzen können, während Sie bei anderen Methoden, wie Entscheidungsbäumen und Sup-

e-bol.net

port-Vektor-Maschinen, die gesamte Datenmenge auf einmal benötigen.) Diese Unterstützung des inkrementellen Trainings ist für Anwendungen wie Spam-Filter sehr wichtig, weil sie fortlaufend durch neu eintreffende E-Mails trainiert werden, schnell aktualisiert werden müssen und eventuell gar keinen Zugriff auf alle bisher empfangenen E-Mails haben.

Ein anderer großer Vorteil des naiven Bayes-Klassifizierers ist, dass man recht einfach interpretieren kann, was der Klassifizierer gelernt hat. Da die Wahrscheinlichkeiten jedes Merkmals gespeichert werden, können Sie sich jederzeit Ihre Datenbank anschauen und erkennen, welche Merkmale am meisten zur Unterscheidung zwischen Spam und erwünschter Mail oder Programmiersprachen und Schlangen beitragen. Diese Information ist interessant und kann eventuell auch für andere Anwendungen genutzt werden oder als Ausgangsbasis für Anwendungen dienen.

Der größte Nachteil der naiven Bayes-Klassifizierer ist, dass sie nicht mit Ergebnissen umgehen können, die sich aufgrund von Merkmalskombinationen verändern. Stellen Sie sich ein Szenario vor, in dem Sie versuchen, Spam-Mails von erwünschten Mails zu trennen: Nehmen wir an, Sie bauen Webanwendungen. Daher taucht das Wort »online« häufig in Ihren arbeitsbezogenen Mails auf. Ihr bester Freund arbeitet in einer Apotheke und schickt Ihnen gern lustige Geschichten über Dinge, die bei seiner Arbeit passiert sind. Zudem erhalten Sie – wie viele andere Personen, die nicht sorgsam auf ihre E-Mail-Adressen geachtet haben – gelegentlich Spam, der die Wörter »online pharmacy« enthält.

Sie sehen das Problem hier vermutlich schon – dem Klassifizierer wird regelmäßig mitgeteilt, dass »online« und »pharmacy« in erwünschten Mails enthalten sind, daher sind ihre Wahrscheinlichkeiten dafür recht hoch. Teilen Sie dem Klassifizierer nun mit, dass eine bestimmte E-Mail mit den Wörtern »online pharmacy« Spam ist, werden diese Wörter wieder ein wenig in Richtung Spam hin angepasst, wodurch ein fortdauernder Richtungsstreit entsteht. Da die Merkmale separate Wahrscheinlichkeiten haben, kann der Klassifizierer nichts zu Kombinationen sagen. Bei der Klassifikation von Dokumenten ist das meist nicht so schlimm, da eine E-Mail mit den Wörtern »online pharmacy« vermutlich auch noch andere Anzeichen für Spam enthält, aber bei anderen Problemen kann es deutlich wichtiger sein, Merkmalskombinationen erkennen zu können.

Entscheidungsbaum-Klassifizierer

Entscheidungsbäume wurden in Kapitel 7 vorgestellt, um Ihnen zu zeigen, wie Sie ein Modell zum Benutzerverhalten aus Serverlogs aufbauen können. Entscheidungsbäume sind bekannt dafür, besonders leicht verständlich und interpretierbar zu sein. Ein Beispiel für einen Entscheidungsbaum finden Sie in Abbildung 12-1.

e-bol.net

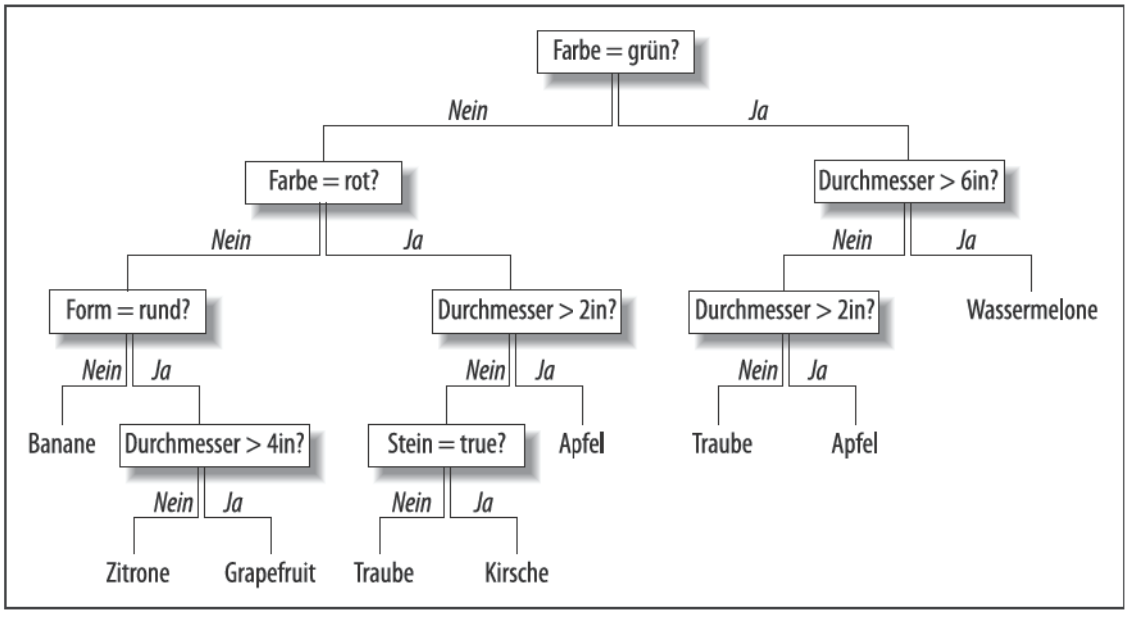

Abbildung 12-1: Beispiel-Entscheidungsbaum

Aus der Abbildung sollte klar sein, was ein Entscheidungsbaum tut, wenn er ein neues Element klassifizieren soll. Beginnend mit dem Knoten an der Wurzel des Baums, prüft er die Kriterien des Knotens für das Element – wenn es passt, folgt er dem Ja-Zweig, ansonsten dem Nein-Zweig. Dieser Prozess wiederholt sich, bis ein Endpunkt erreicht ist, an dem man die vorhergesagte Kategorie findet.

Training

Das Klassifizieren in einem Entscheidungsbaum ist ziemlich einfach, das Trainieren dagegen umso aufwendiger. Der in Kapitel 7 beschriebene Algorithmus erstellt den Baum von der Wurzel aus, wobei er bei jedem Schritt ein Attribut auswählt, das die Daten so gut wie möglich aufteilt. Um das deutlich zu machen, schauen Sie sich die Früchte-Datenmenge in Tabelle 12-3 an. Dies wird als ursprüngliche Menge bezeichnet.

Tabelle 12-3: Fruchtdaten

Durchmesser	Farbe	Frucht
4	Rot	Apfel
4	Grün	Apfel
1	Rot	Kirsche
1	Grün	Traube
5	Rot	Apfel

e-bol.net

Es gibt zwei mögliche Variablen, anhand deren sich die Daten aufteilen lassen – entweder den Durchmesser oder die Farbe. Im ersten Schritt werden beide Variablen ausprobiert, um entscheiden zu können, welche dieser Variablen die Daten am besten unterteilen. Nimmt man diese Aufteilung anhand der Farbe vor, erhält man das Ergebnis aus Tabelle 12-4.

Tabelle 12-4: Fruchtdaten unterteilt nach der Farbe

Rot	Grün
Apfel	Apfel
Kirsche	Traube
Apfel	

Die Daten sind immer noch ziemlich gemischt. Teilen Sie sie aber anhand des Durchmessers auf (kleiner als 4 Inch beziehungsweise größer oder gleich 4 Inch), ist die Aufteilung deutlich sauberer (bezeichnet als Subset1 auf der rechten und Subset2 auf der linken Seite). Diese Unterteilung ist in Tabelle 12-5 zu sehen.

Tabelle 12-5: Fruchtdaten unterteilt nach dem Durchmesser

Durchmesser < 4 in	Durchmesser ≥ 4 in
Kirsche	Apfel
Traube	Apfel

Das ist ganz offensichtlich das deutlich bessere Ergebnis, da Subset1 alle Äpfel-Einträge aus der ursprünglichen Datenmenge enthält. Natürlich ist die bessere Variable in diesem Beispiel leicht zu finden, aber bei größeren Datenmengen wird es nicht immer solche leichten Entscheidungen geben. Kapitel 7 stellt das Konzept der Entropie vor (der Menge an Unordnung in einem Set), um messen zu können, wie gut eine Aufteilung ist:

- *p(i) = frequency(Ergebnis) = count(Ergebnis) / count(Datensätze)*
- *Entropie = Summe p(i) * log(p(i)) für alle Ergebnisse*

Eine niedrige Entropie für ein Set besagt damit, dass das Set recht homogen ist. Ein Wert von 0 bedeutet, dass es aus genau einem Typ besteht. Subset1 (Durchmesser ≥ 4) in Tabelle 12-5 hat eine Entropie von 0. Die Entropie jedes Sets wird genutzt, um den *Informationsgewinn* zu ermittelt, der wie folgt definiert ist:

- *weight1 = Größe von Subset1 / Größe des Ursprungssets*
- *weight2 = Größe von Subset2 / Größe des Ursprungssets*
- *Gewinn = Entropie(Original) – weight1*Entropie(Set1) – weight2*Entropie(Set2)*

e-bol.net

So wird für jede mögliche Aufteilung der Informationsgewinn berechnet und genutzt, um die Teilungsvariable zu bestimmen. Nachdem die Teilungsvariable ausgewählt wurde, kann der erste Knoten angelegt werden (siehe Abbildung 12-2).

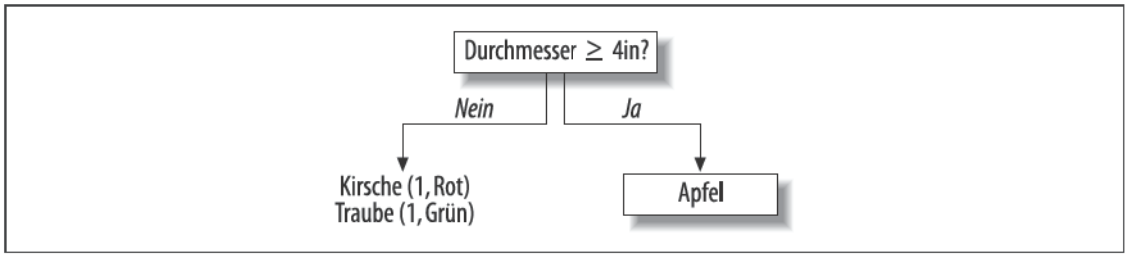

Abbildung 12-2: Wurzelknoten des Entscheidungsbaums für Früchte

Die Kriterien werden beim Knoten angezeigt. Die Daten, die diesen Kriterien nicht entsprechen, werden den Nein-Zweig entlang weitergeleitet, die anderen den Ja-Zweig. Da der Ja-Zweig jetzt nur noch ein mögliches Ergebnis bietet, wird er zu einem Endpunkt. Der Nein-Zweig bietet immer noch eine Mischung, daher lässt er sich mit der eben angewandten Methode weiter unterteilen. In diesem Fall ist die Farbe die beste Variable, nach der die Daten unterteilt werden können. Dieser Prozess wiederholt sich, bis es durch das Aufteilen eines gegebenen Zweigs keinen Informationsgewinn mehr gibt.

Verwenden Ihres Entscheidungsbaums

Der Code für die Entscheidungsbäume aus Kapitel 7 wurde mit einer Liste aus Listen trainiert. Jede interne Liste besaß eine Reihe von Werten, wobei der letzte Wert in der Liste die Kategorie war. Sie können die einfache Frucht-Datenmenge so anlegen:

```
>>> fruit=[[4,'Rot','Apfel'],
... [4,'Grün','Apfel'],
... [1,'Rot','Kirsche'],
... [1,'Grün','Traube'],
... [5,'Rot','Apfel']]
```

Jetzt können Sie den Entscheidungsbaum trainieren und ihn nutzen, um neue Beispiele zu klassifizieren:

```
>>> import treepredict
>>> tree=treepredict.buildtree(fruit)
>>> treepredict.classify([2,'Rot'],tree)
{'Kirsche': 1}
>>> treepredict.classify([5,'Rot'],tree)
{'Apfel': 3}
>>> treepredict.classify([1,'Grün'],tree)
{'Traube': 1}
>>> treepredict.classify([120,'Lila'],tree)
{'Apfel': 3}
```

e-bol.net

Klar ist etwas mit zehn Fuß Durchmesser in Lila kein Apfel, aber der Entscheidungsbaum ist auf das beschränkt, was er schon kennt. Schließlich können Sie den Baum ausdrucken oder als Grafik darstellen, um den Prozess bei der Entscheidungsfindung zu verstehen:

```
>>> treepredict.printtree(tree)
0:4?
T-> {'Apfel': 3}
F-> 1:Grün?
  T-> {'Traube': 1}
  F-> {'Kirsche': 1}
```

Stärken und Schwächen

Der größte Vorteil von Entscheidungsbäumen ist das einfache Interpretieren eines trainierten Modells. Zudem bringen die Algorithmen die wichtigen Faktoren an die Spitze des Baums. Das bedeutet, dass ein Entscheidungsbaum nicht nur zur Klassifikation, sondern auch zur Interpretation nützlich ist. Wie beim Bayes-Klassifizierer können Sie unter die Motorhaube schauen und verstehen, warum der Baum so entscheidet, wie er es tut. Das kann Ihnen auch bei Entscheidungen helfen, die außerhalb des Klassifikationsprozesses liegen. So hat zum Beispiel das Modell in Kapitel 7 vorhergesagt, welche Benutzer zu zahlenden Kunden werden. Durch den Entscheidungsbaum, der zeigt, welche Variablen die Daten am besten aufteilen, kann man eine Werbestrategie planen oder auch entscheiden, welche Daten in Zukunft gesammelt werden sollen.

Entscheidungsbäume funktionieren auch mit numerischen Daten als Eingabewerten, da sie die Trennlinie finden, die den Informationsgewinn maximiert. Die Fähigkeit, Kategoriedaten und numerische Daten zu mischen, ist bei vielen Problemarten nützlich – klassische statistische Methoden wie die Regression haben dabei Probleme. Andererseits sind Entscheidungsbäume beim Vorhersagen numerischer Ergebnisse nicht so geschickt. Ein Regressionsbaum kann die Daten in Durchschnittswerte mit der kleinsten Varianz aufteilen, aber wenn die Daten komplex sind, wird der Baum sehr groß werden, bevor er genaue Entscheidungen treffen kann.

Der größte Vorteil von Entscheidungsbäumen gegenüber dem Bayes-Klassifizierer ist, dass sie gut mit Verbindungen zwischen Variablen klarkommen. Ein Spam-Filter, der auf einem Entscheidungsbaum basiert, würde schnell erkennen, dass »online« und »pharmacy« einzeln genehm sind, zusammen aber Spam deutlich machen.

Leider ist der Algorithmus aus Kapitel 7 für einen Spam-Filter nicht sehr nützlich, da er kein inkrementelles Training unterstützt. (Alternative Algorithmen für Entscheidungsbäume, die ein inkrementelles Training ermöglichen, sind zurzeit Gegenstand der Forschung.) Sie können viele Dokumente nehmen und einen Ent-

e-bol.net

scheidungsbaum zum Filtern von Spam aufbauen, aber er lässt sich nicht mit jeder neuen E-Mail anpassen – Sie müssten immer von vorne beginnen. Da viele Menschen Zehntausende von E-Mails haben, wäre das unpraktisch. Da zudem die Anzahl möglicher Knoten sehr groß wäre (jedes Merkmal ist anwesend oder abwesend), könnten die Bäume extrem groß und komplex werden, wodurch die Klassifikation sehr langsam würde.

Neuronale Netze

Kapitel 4 hat gezeigt, wie man ein einfaches neuronales Netz zum Anpassen der Bewertungen von Suchergebnissen aufbauen kann, das darauf basiert, welche Links die Benutzer in der Vergangenheit angeklickt haben. Dieses neuronale Netz konnte lernen, welche Wörter in welchen Kombinationen wichtig und welche Wörter bei einer bestimmten Abfrage unwichtig waren. Neuronale Netze können sowohl auf Klassifikationsprobleme wie auch auf Probleme beim Vorhersagen numerischer Werte angewandt werden.

Das neuronale Netz in Kapitel 4 wurde als Klassifizierer verwendet – es lieferte eine Zahl für jeden Link, wobei vorhergesagt wurde, dass der Link mit der höchsten Zahl derjenige sein würde, den der Benutzer anklickt. Da es Zahlen für jeden Link gab, konnten Sie damit die Bewertung der Suchergebnisse vornehmen.

Es gibt viele verschiedene Arten neuronaler Netze. Das in diesem Buch behandelte ist als mehrlagiges Perzeptron-Netz bekannt, das so benannt wurde, weil es eine Schicht mit Eingabeneuronen besitzt, die eine oder mehrere Schichten verborgener Neuronen ansteuert. Die prinzipielle Struktur ist in Abbildung 12-3 zu sehen.

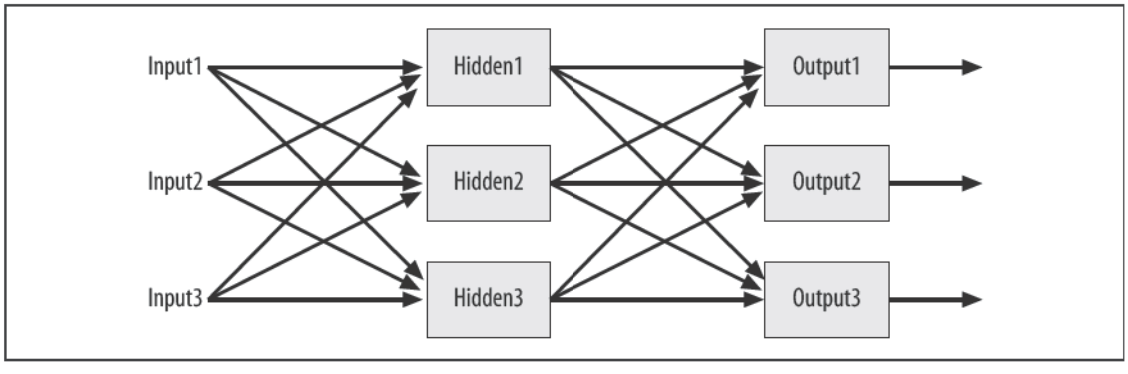

Abbildung 12-3: Prinzipielle Struktur eines neuronalen Netzes

Dieses Netz hat zwei Neuronenschichten. Die Schichten sind miteinander durch *Synapsen* verbunden, die jeweils eine Gewichtung besitzen. Je höher die Gewichtung einer Synapse von einem Neuron zum nächsten ist, desto mehr Einfluss hat sie auf die Ausgabe des Neurons.

e-bol.net

Stellen Sie sich als einfaches Beispiel nochmals das Filtern von Spam vor, das im weiter oben befindlichen Abschnitt »Bayes-Klassifizierer« beschrieben wird. In unserer vereinfachten E-Mail-Welt kann eine E-Mail das Wort »online«, das Wort »pharmacy« oder beide Wörter enthalten. Um herauszufinden, welche der Nachrichten Spam sind, könnten Sie ein neuronales Netz nutzen, das so aussieht wie das in Abbildung 12-4.

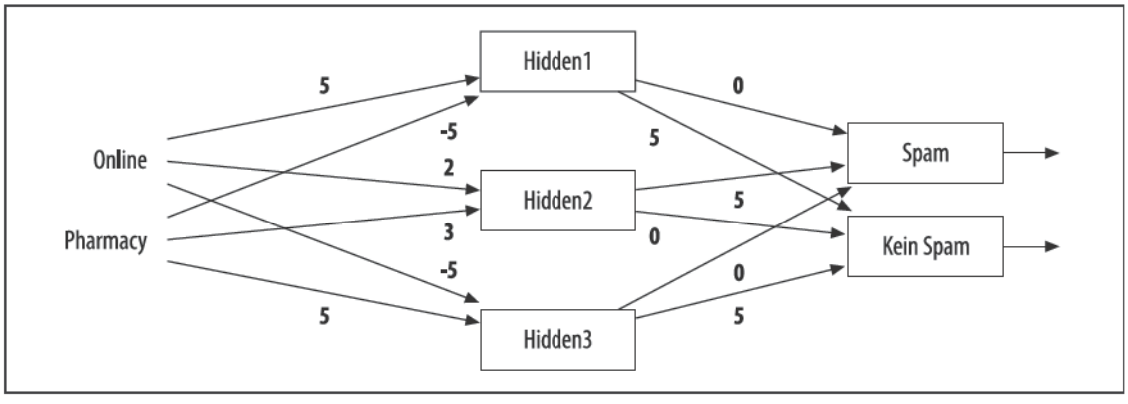

Abbildung 12-4: Neuronales Netz zur Spam-Klassifikation

In dieser Abbildung sind die Gewichtungen der Synapsen schon vorgegeben, um das Problem zu lösen. (Sie werden im nächsten Abschnitt erfahren, wie diese Gewichtungen gesetzt werden.) Die Neuronen in der ersten Schicht reagieren auf die Wörter, die als Eingabe genutzt werden – wenn ein Wort in der E-Mail vorhanden ist, werden die Neuronen aktiv, die stark mit diesem Wort verbunden sind. Die zweite Schicht wird durch die erste befeuert und reagiert damit auf Wortkombinationen. Schließlich liefern diese Neuronen ihre Ergebnisse als Ausgabe, wobei bestimmte Kombinationen mehr oder weniger stark mit den möglichen Ergebnissen verbunden sind. Am Ende gewinnt die Entscheidung, deren Ausgabe am stärksten ist. Abbildung 12-5 zeigt, wie das Netz auf das Wort »online« reagiert, wenn es *nicht* vom Wort »pharmacy« begleitet wird.

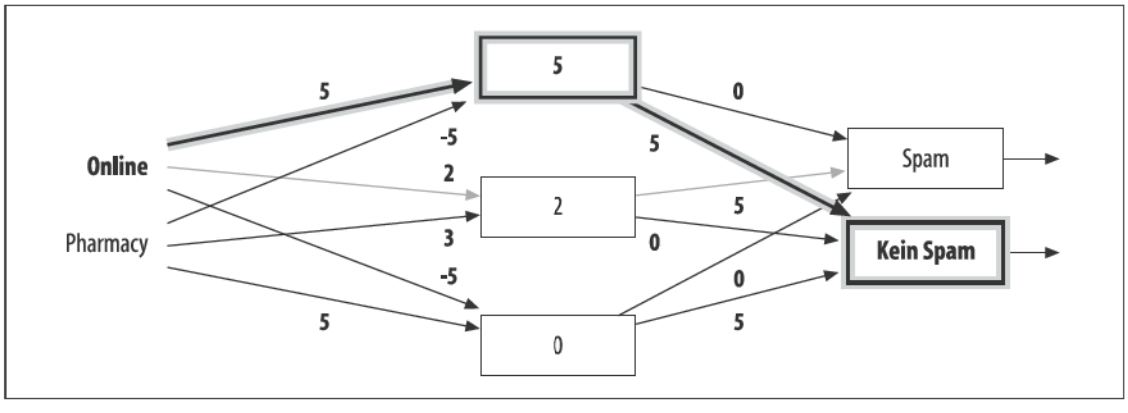

Abbildung 12-5: Reaktion des Netzes auf das Wort »online«

e-bol.net

Eines der Neuronen aus der ersten Schicht reagiert auf »online« und übergibt seine Ausgabe an die zweite Schicht, in der ein Neuron gelernt hat, Nachrichten zu erkennen, die nur das Wort »online« enthalten. Dieses Neuron hat eine deutlich stärkere Synapse zu »Kein Spam« als zu »Spam«, daher wird die Nachricht als »Kein Spam« klassifiziert. Abbildung 12-6 zeigt, was geschieht, wenn die Wörter »online« und »pharmacy« gemeinsam an das Netz übergeben werden.

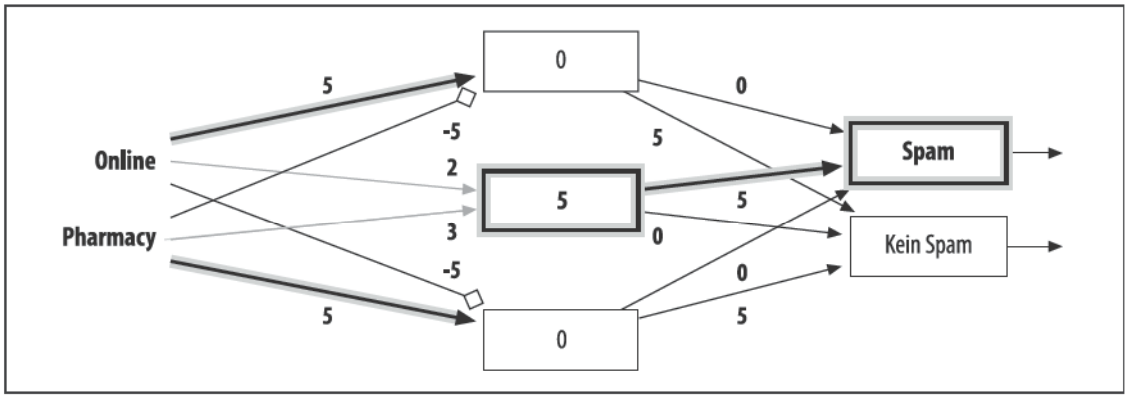

Abbildung 12-6: Neuronales Netz reagiert auf »online pharmacy«

Da die Neuronen in der ersten Schicht auf die einzelnen Wörter reagieren, werden sie beide aktiviert. In der zweiten Schicht ist es ein bisschen interessanter. Die Anwesenheit von »pharmacy« wirkt sich negativ auf das Neuron aus, das für »online« allein steht, während beide Neuronen der ersten Schicht auf das mittlere Neuron einwirken, das darauf trainiert wurde, auf »online« und »pharmacy« zusammen zu reagieren. Dieses Neuron wirkt sich sehr stark auf die »Spam«-Kategorie aus, daher wird das Dokument als Spam klassifiziert. Dieses Beispiel zeigt, wie mehrlagige neuronale Netze gut mit verschiedenen Merkmalen umgehen können, die je nach Kombination etwas anderes bedeuten.

Trainieren eines neuronalen Netzes

Im obigen Beispiel hat das neuronale Netz schon die passenden Gewichtungen für alle Synapsen. Das wirklich Spannende bei neuronalen Netzen ist ihre Fähigkeit, mit Zufallsgewichtungen beginnen zu können und dann durch Training aus Beispielen zu lernen. Die gebräuchlichste Methode beim Trainieren eines mehrlagigen Perzeptron-Netzes wird als Backpropagation bezeichnet und ist ebenfalls in Kapitel 4 beschrieben.

Um ein Netz per Backpropagation zu trainieren, beginnen Sie mit einem Beispiel wie dem Wort »online« selbst sowie der richtigen Antwort, die in diesem Fall »Kein Spam« lautet. Sie füttern nun das Netz mit dem Beispiel und prüfen, wie die Kategorisierung aussieht.

e-bol.net

Zunächst wird das Netz vielleicht ein etwas größeres Ergebnis für »Spam« als für »Kein Spam« liefern, was falsch ist. Um das zu korrigieren, wird ihm mitgeteilt, dass »Spam« näher an 0 und »Kein Spam« näher an 1 liegen sollte. Die Synapsengewichtungen, die zu »Spam« führen, werden proportional zu ihrem Beitrag ein wenig reduziert und die Gewichtungen für »Kein Spam« entsprechend verstärkt. Die Synapsengewichtungen zwischen der Eingabeschicht und den verborgenen Schichten werden ebenfalls entsprechend ihrem Beitrag zu den wichtigen Knoten in der Ausgabeschicht verändert.

Die eigentliche Formel für diese Anpassungen wird in Kapitel 4 vorgestellt. Um das Netz davon abzuhalten, durch ein Training mit verrauschten oder ungenauen Daten überkompensiert zu werden, wird es langsam trainiert, sodass es beim Klassifizieren eines bestimmten Beispiels umso besser wird, je häufiger es dieses zu sehen bekommt.

Verwenden Ihres Codes für das neuronale Netz

Der Code in Kapitel 4 lässt sich einfach auf dieses Problem anwenden, man darf nur nicht die Wörter direkt verwenden. Arbeiten Sie mit numerischen IDs für alles, sodass Sie als mögliche Eingaben Zahlen nutzen. Es wird eine Datenbank verwendet, um die Trainingsdaten zu speichern, daher öffnen Sie sie einfach mit einem Dateinamen und beginnen mit dem Training:

```
>>> import nn
>>> online,pharmacy=1,2
>>> spam,notspam=1,2
>>> possible=[spam,notspam]
>>> neuralnet=nn.searchnet('nntest.db')
>>> neuralnet.maketables()
>>> neuralnet.trainquery([online],possible,notspam)
>>> neuralnet.trainquery([online,pharmacy],possible,spam)
>>> neuralnet.trainquery([pharmacy],possible,notspam)
>>> neuralnet.getresult([online,pharmacy],possible)
[0.7763, 0.2890]
>>> neuralnet.getresult([online],possible)
[0.4351, 0.1826]
>>> neuralnet.trainquery([online],possible,notspam)
>>> neuralnet.getresult([online],possible)
[0.3219, 0.5329]
>>> neuralnet.trainquery([online],possible,notspam)
>>> neuralnet.getresult([online],possible)
[0.2206, 0.6453]
```

Sie können erkennen, dass das Netz umso eindeutigere Ergebnisse liefert, je mehr es trainiert ist. Es kann auch mit gelegentlichen falschen Beispielen umgehen und trotzdem eine gute Vorhersage liefern.

e-bol.net

Stärken und Schwächen

Die größte Stärke von neuronalen Netzen ist der Umgang mit komplexen, nicht linearen Funktionen und die Möglichkeit, Abhängigkeiten zwischen unterschiedlichen Eingabewerten zu erkennen. Auch wenn im Beispiel nur die Eingabewerte 1 oder 0 (vorhanden oder abwesend) genutzt wurden, kann jede Zahl als Eingabe verwendet werden, und das Netz kann Zahlen als Ausgabewerte vorhersagen.

Neuronale Netze ermöglichen auch inkrementelles Training und erfordern nicht viel Platz, um die trainierten Modelle abzuspeichern, da es sich dabei nur um Listen mit Zahlen handelt, die die Synapsengewichtungen darstellen. Es ist nicht notwendig, die Ursprungsdaten nach dem Training weiter aufzubewahren, was bedeutet, dass neuronale Netze für Anwendungen genutzt werden können, in denen es einen kontinuierlichen Fluss von Trainingsdaten gibt.

Der größte Nachteil neuronaler Netze ist, dass es sich bei ihnen um eine Blackbox-Methode handelt. Das hier gezeigte Beispielnetz war so aufgebaut, dass man es sehr leicht verstehen konnte, aber echte Netze können Hunderte von Knoten und Tausende von Synapsen enthalten, wodurch es unmöglich wird herauszufinden, wie das Netz zu der von ihm gelieferten Antwort kam. Für manche Anwendungen kann es jedoch inakzeptabel sein, den Begründungsprozess nicht zu verstehen.

Ein anderer Nachteil ist, dass es keine festen Regeln für die Wahl der Trainingsrate und der Netzgröße für ein bestimmtes Problem gibt. Diese Entscheidung erfordert im Allgemeinen viel Erfahrung. Wählt man eine zu große Trainingsrate, kann dieses zu einer Übergeneralisierung des Netzes in Bezug auf verrauschte Daten führen – ist die Rate zu klein, kann es sein, dass das Netz mit Ihren Daten niemals richtig »lernt«.

Support-Vektor-Maschinen

Support-Vektor-Maschinen (SVMs) wurden in Kapitel 9 vorgestellt und stellen vermutlich die komplexesten Klassifikationsmethoden dar, die in diesem Buch behandelt wurden. SVMs erwarten Datenmengen mit numerischen Eingaben und versuchen vorherzusagen, in welche Kategorie sie fallen. Vielleicht wollen Sie für ein Basketballteam auf Basis einer Liste mit Größen und Laufgeschwindigkeiten der einzelnen Mitspieler ermitteln, wer wo spielen soll. Zur Vereinfachung nehmen wir nur zwei Möglichkeiten – einen Angreifer, für den lange Spieler benötigt werden, und einen Verteidiger, der vor allem schnell laufen können muss.

Eine SVM erstellt ein Vorhersagemodell, indem sie die Trennlinie zwischen den beiden Kategorien ermittelt. Wenn Sie die Werte für die Höhe gegen die Geschwindigkeitswerte sowie die beste Position für jede Person eintragen, erhalten Sie einen Graphen, der so aussieht wie der in Abbildung 12-7. Angriffsspieler werden als X, Verteidiger als O dargestellt. Zudem sieht man im Graphen ein paar Linien, die die Daten in zwei Kategorien aufteilen.

e-bol.net

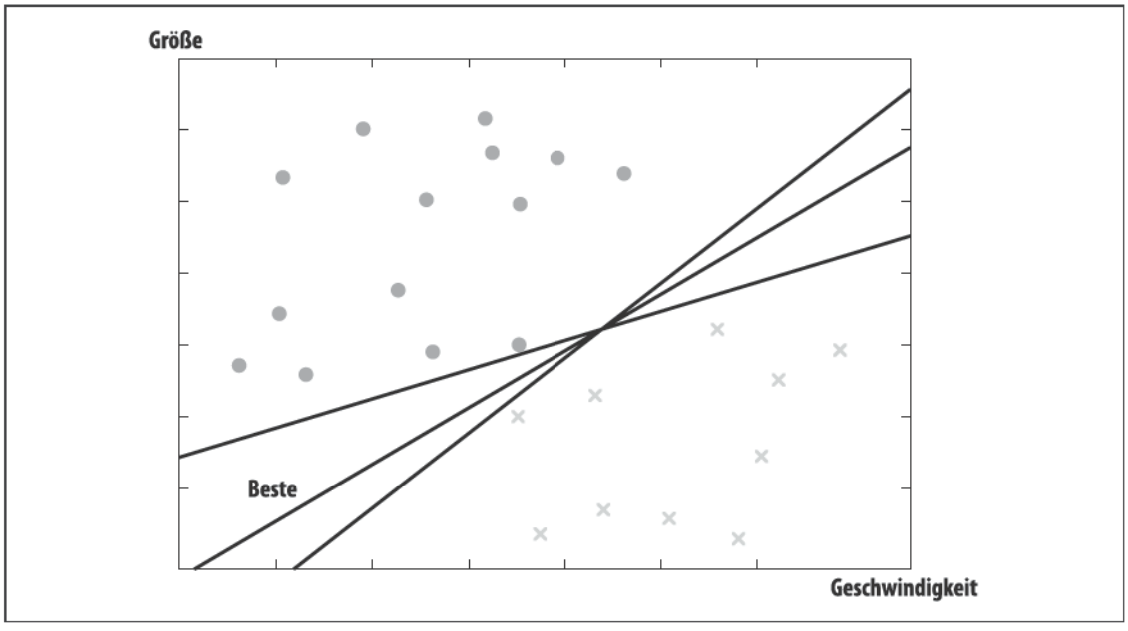

Abbildung 12-7: Basketballspieler und Trennlinien

Eine Support-Vektor-Maschine findet die Linie, die die Daten am saubersten trennt. Das bedeutet, diese Linie hat den größtmöglichen Abstand zu den Punkten, die in der Nähe der Linie liegen. Obwohl in Abbildung 12-7 alle Linien die Daten aufteilen, erledigt es diejenige mit der Bezeichnung »Beste« am optimalsten. Die einzigen Punkte, die zu ihrer Ermittlung notwendig sind, sind die ihr am nächsten liegenden – sie werden als *Support-Vektoren* bezeichnet.

Nachdem die Trennlinie gefunden wurde, geht es beim Klassifizieren neuer Elemente nur noch darum, sie im Graphen einzutragen und zu schauen, auf welcher Seite der Linie sie liegen. Man muss die Trainingsdaten nicht erneut durchlaufen, wenn man neue Punkte klassifizieren will, daher ist die Klassifikation sehr schnell.

Der Kernel-Trick

Support-Vektor-Maschinen nutzen ebenso wie andere lineare Klassifizierer, die auf dem *Skalarprodukt* für Vektoren basieren, häufig eine Technik namens *Kernel-Trick*. Um das zu verstehen, stellen Sie sich vor, wie sich das Problem ändern würde, wenn die vorherzusagende Klassifikation nicht die Position wäre, sondern die Entscheidung darüber, ob die Spieler in ein Amateurteam passen würden, in dem die Positionen häufig gewechselt werden. Das ist interessanter, weil die Aufteilung nicht linear ist. Sie wollen keine Spieler, die zu lang oder zu schnell sind, da das Spiel für die anderen damit zu schwierig werden würde, aber sie sollen auch nicht zu klein oder zu langsam sein. Abbildung 12-8 zeigt, wie das aussehen könnte, wobei ein O angibt, dass ein Spieler für das Team passend ist, und ein X, dass er nicht passt.

e-bol.net

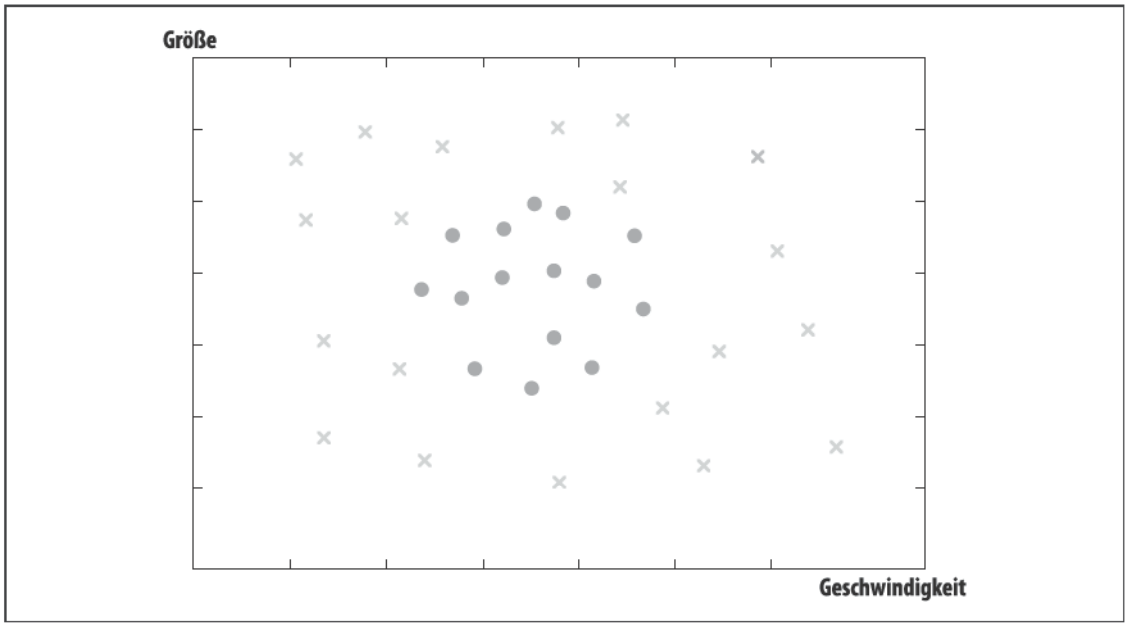

Abbildung 12-8: Graph für Basketballspieler in einem Amateurteam

Es gibt keine gerade Trennlinie, die eine saubere Aufteilung ermöglicht, daher können Sie keinen linearen Klassifizierer nutzen, um die Trennlinie zu ziehen, ohne die Daten vorher irgendwie zu verändern. Eine Möglichkeit wäre, die Daten in einen anderen Raum zu transformieren – vielleicht in einen mit mehr als zwei Dimensionen –, indem verschiedene Funktionen auf die Achsenvariablen angewendet werden. In diesem Fall könnten wir zum Beispiel einen neuen Raum erzeugen, indem wir die Durchschnittswerte für die Größe und die Geschwindigkeit abziehen und die verbleibenden Werte quadrieren. Das würde so aussehen wie in Abbildung 12-9.

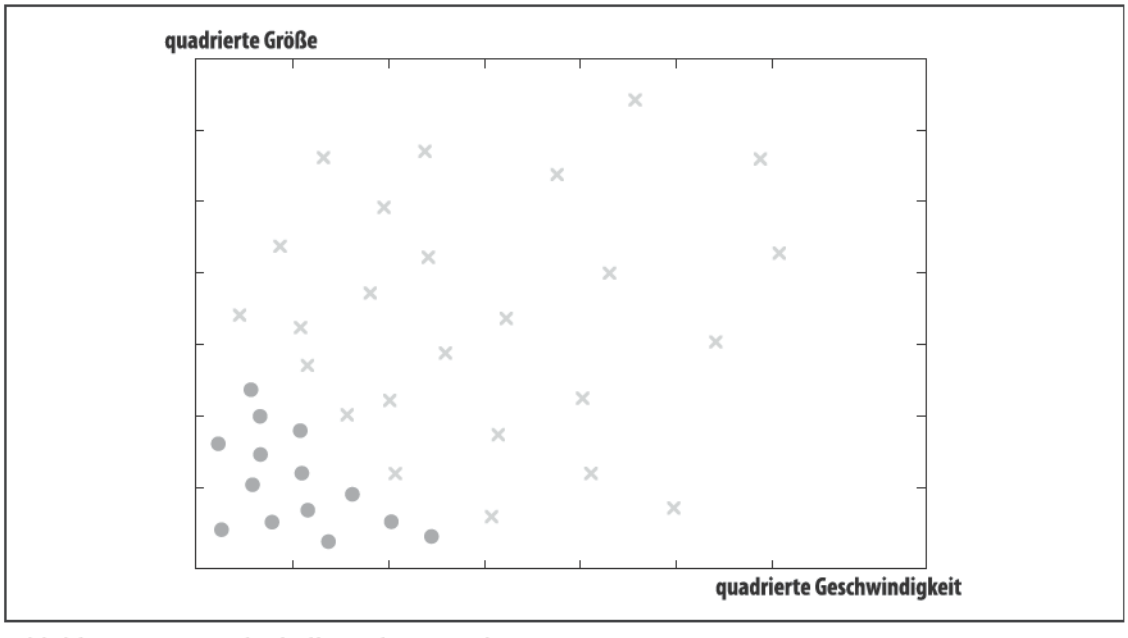

Abbildung 12-9: Basketball-Spieler im Polynom-Raum

e-bol.net

Dies wird als *Polynom-Transformation* bezeichnet und wandelt Daten auf andere Achsen um. Jetzt erkennt man direkt, dass es eine Trennlinie zwischen den passenden und nicht passenden Spielern für das Team gibt, die mit einem linearen Klassifizierer gefunden werden kann. Zum Klassifizieren neuer Punkte muss man diese nur in diesen Raum transformieren und schauen, auf welche Seite der Linie sie fallen.

Die Transformation funktioniert in diesem Fall, aber in vielen anderen würde das Finden der Trennlinie eine Transformation in deutlich komplexere Räume erfordern. Manche dieser Räume haben Tausende oder sogar unendlich viele Dimensionen, daher ist es nicht immer praktikabel, diese Transformation auch wirklich durchzuführen. Hier kommt der Kernel-Trick ins Spiel – anstatt den Raum zu transformieren, ersetzen Sie die Skalarproduktfunktion durch eine Funktion, die das zurückgibt, was das Skalarprodukt liefern *würde*, wenn die Daten in einen anderen Raum transformiert worden wären. Anstatt zum Beispiel die obige Polynom-Transformation durchzuführen, würden Sie

```
dotproduct(A,B)
```

ändern in:

```
dotproduct(A,B)**2
```

In Kapitel 9 haben Sie einen einfachen linearen Klassifizierer erstellt, der Gruppenmittel genutzt hat. Sie haben gesehen, wie dies geändert werden kann, indem die Skalarproduktfunktion durch andere Funktionen zum Kombinieren von Vektoren ersetzt wird, wodurch auch nicht lineare Probleme gelöst werden können.

LIBSVM verwenden

Kapitel 9 hat eine Bibliothek namens LIBSVM vorgestellt. Sie können sie nutzen, um mit einer Datenmenge zu trainieren (zum Finden der Trennlinie in einem transformierten Raum) und um neue Beobachtungen zu klassifizieren:

```
>>> from random import randint
>>> # 200 zufällige Punkte erstellen.
>>> d1=[[randint(-20,20),randint(-20,20)] for i in range(200)]
>>> # Mit 1 klassifizieren, wenn sie im Kreis sind, sonst mit 0.
>>> result=[(x**2+y**2)<144 and 1 or 0 for (x,y) in d1]
>>> from svm import *
>>> prob=svm_problem(result,d1)
>>> param=svm_parameter(kernel_type=RBF)
>>> m=svm_model(prob,param)
>>> m.predict([2,2])
1.0
>>> m.predict([14,13])
0.0
>>> m.predict([-18,0])
0.0
```

e-bol.net

LIBSVM unterstützt viele verschiedene Kernel-Funktionen, und es ist einfach, sie durch verschiedene Parameter auszuprobieren, um herauszufinden, was für eine gegebene Datenmenge am besten funktioniert. Um zu testen, wie gut ein Modell arbeitet, können Sie die Funktion `cross_validation` nutzen, die einen Parameter `n` nimmt, der die Datenmenge in *n* Untersets aufteilt. Dann nutzt sie jede Untermenge als Testset und trainiert das Modell mit allen anderen Untermengen. Sie liefert eine Liste mit Antworten zurück, die Sie mit der ursprünglichen Liste vergleichen können:

```
>>> guesses=cross_validation(prob,param,4)
>>> sum([abs(guesses[i]-result[i]) for i in range(len(guesses))])
28.0
```

Wenn Sie ein neues Problem angehen, können Sie die verschiedenen Kernel-Funktionen mit unterschiedlichen Parametern austesten, um die besten Ergebnisse zu ermitteln. Das hängt stark von der Datenmenge ab, und nachdem Sie sich entschieden haben, können Sie diese Parameter nutzen, um das Modell zu erstellen, das zum Klassifizieren neuer Beobachtungen genutzt wird. In der Realität können Sie ein paar verschachtelte Schleifen erstellen, um verschiedene Werte auszutesten und zu beobachten, welche die besten Ergebnisse liefern.

Stärken und Schwächen

Support-Vektor-Maschinen sind sehr mächtige Klassifizierer – sobald Sie einmal korrekte Parameter haben, arbeiten sie so gut wie jede andere Klassifikationsmethode, die in diesem Buch behandelt wurde – oder sogar besser. Zudem sind sie nach dem Training sehr schnell beim Klassifizieren neuer Beobachtungen, da man dabei nur bestimmen muss, auf welcher Seite einer Trennlinie sich der Punkt befindet. Indem Kategoriedaten in Zahlen umgewandelt werden, können Sie auch mit numerischen und kategorischen Daten arbeiten.

Ein Nachteil ist, dass die besten Kernel-Transformationsfunktion und die Parameter für diese Funktion für jede Datenmenge ein wenig anders sein werden und Sie diese jedes Mal herausfinden müssen. Man kann zwar mögliche Werte durchspielen, um dieses Problem zu beheben, aber trotzdem brauchen Sie dafür eine Datenmenge, die groß genug ist, um eine vernünftige Kreuzvalidierung durchzuführen. Im Allgemeinen funktionieren SVMs besser, wenn für das Problem viele Daten vorhanden sind, während andere Methoden wie zum Beispiel Entscheidungsbäume auch bei sehr kleinen Datenmengen noch interessante Informationen liefern können.

Wie neuronale Netze sind SVMs eine Blackbox – es ist aufgrund der Transformationen in höherdimensionale Räume sogar noch schwieriger, den Klassifikationsprozess eines SVM zu interpretieren. Ein SVM mag wunderbare Antworten liefern, aber Sie werden nie herausfinden, warum.

e-bol.net

k-nächste Nachbarn

Kapitel 8 behandelte die numerische Vorhersage mit einem Algorithmus namens k-nächste Nachbarn (kNN) und nutzte ihn, um zu zeigen, wie Sie Modelle zur Preisvorhersage bei einem Set gegebener Beispiele aufbauen können. Der Empfehlungsalgorithmus in Kapitel 2 für die Vorhersage, wie sehr jemand einen Film oder einen Link mögen würde, war ebenfalls eine einfachere Version von kNN.

kNN erwartet ein neues Element, für das Sie eine numerische Vorhersage brauchen, und vergleicht es mit einer Reihe von Elementen, für die es bereits Werte hat. Der Algorithmus findet die ähnlichsten Elemente und mittelt ihre Werte, um eine Schätzung abgeben zu können. Tabelle 12-6 zeigt eine Liste mit Digitalkameras zusammen mit der Megapixel-Anzahl, dem Zoom und dem Verkaufspreis.

Tabelle 12-6: Digitalkameras und Preise

Kamera	Megapixel	Zoom	Preis
C1	7,1	3,8x	$399
C2	5,0	2,4x	$299
C3	6,0	4,0x	$349
C4	6,0	12,0x	$399
C5	10,0	3x	$449

Angenommen, Sie möchten den Preis für eine neue Kamera mit sechs Megapixeln und einem 6x-Zoom abschätzen. Als Erstes brauchen Sie eine Möglichkeit zu messen, wie ähnlich sich zwei Elemente sind. In Kapitel 8 wurde der euklidische Abstand genutzt, Sie haben in diesem Buch aber auch schon andere Abstandsmetriken gesehen, wie zum Beispiel die Pearson-Korrelation und den Tanimoto-Score. Unter Verwendung des euklidischen Abstands zeigt dieses Beispiel, dass das naheste Element in der Tabelle C3 ist. Um dies deutlich zu machen, können Sie die Elemente in einen Graphen eintragen, in dem die Megapixel auf der x-Achse und der Zoom auf der y-Achse liegen. Die Elemente selbst werden in Abbildung 12-10 über ihren Preis identifiziert.

Sie könnten nun einfach den Preis $349 als Antwort nehmen (es handelt sich dabei schließlich um den nächsten Nachbarn), aber Sie können nie wissen, ob der Preis nicht ein Ausreißer ist. Aus diesem Grund ist es besser, mehr als den am nächsten liegenden Treffer mit einzubinden und über die Ergebnisse zu mitteln. Das k in k-nächste Nachbarn verweist darauf, wie viele der ähnlichsten Treffer zum Mitteln genommen werden sollen. Wenn Sie zum Beispiel die drei nächsten Nachbarn nutzen und den Durchschnitt bilden, wäre dies der kNN für k=3.

Eine Verbesserung des einfachen Mittelns ist die Verwendung eines gewichteten Durchschnitts, wobei die Gewichtung von der Entfernung des Nachbarn abhängt.

e-bol.net

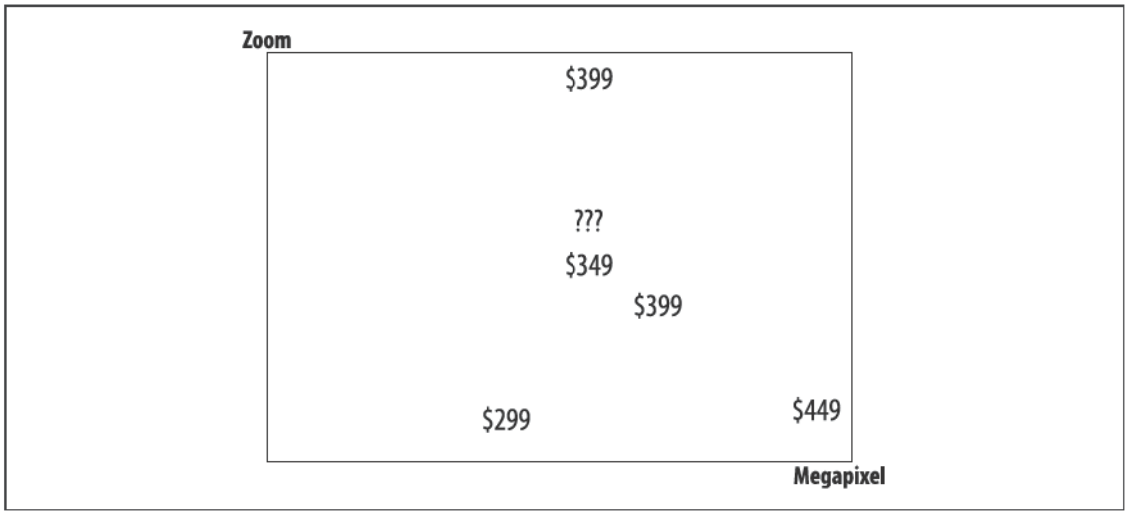

Abbildung 12-10: Kamerapreise im Zoom-Megapixel-Raum

Ein sehr naher Nachbar würde mehr zählen als ein Element, das weiter entfernt ist. Die Gewichtung nimmt dabei mit zunehmendem Abstand ab. Kapitel 8 hat verschiedene Funktionen zum Bestimmen der Gewichte vorgestellt. In diesem Beispiel könnten Sie den Preis von $349 am stärksten gewichten, während die beiden Preise von $399 weniger zählen, zum Beispiel:

```
price = 0.5 * 349 + 0.25 * 399 + 0.25 * 399 = 374
```

Skalieren von Variablen und Ausmerzen überflüssiger Variablen

Ein großes Problem beim bisher beschriebenen kNN-Algorithmus ist, dass er die Entfernungen für alle Variablen berechnet. Wenn die Variablen nun für verschiedene Dinge stehen und eine davon im Allgemeinen deutlich größer ist als die anderen, wird sie auch einen deutlich größeren Einfluss auf die »Nähe« haben. Stellen Sie sich vor, bei der oben angegebenen Datenmenge wäre die Pixelzahl nicht in Megapixeln, sondern in Pixeln angegeben – ein Zoomunterschied von 10x ist deutlich wichtiger als ein Auflösungsunterschied von 10 zusätzlichen Pixeln, aber sie würden gleich behandelt werden. In anderen Fällen wird die Datenmenge Variablen enthalten, die für die Vorhersage völlig nutzlos sind, aber die Abstände trotzdem beeinflussen.

Dieses Problem kann gelöst werden, indem man die Daten skaliert, bevor die Abstände berechnet werden. In Kapitel 8 haben Sie eine Methode zum Skalieren aufgebaut, die die Größe mancher Variablen erhöht und anderer verringert. Komplett unnütze Variablen können mit 0 multipliziert werden, sodass sie das Ergebnis nicht weiter beeinflussen. Variablen, die einen Effekt haben, sich aber in verschiedenen Skalenbereichen befinden, können angepasst werden, damit man sie besser vergleichen kann – so könnten Sie zum Beispiel festlegen, dass ein Unterschied von 2.000 Pixeln einem Zoomunterschied von 1x entspricht.

e-bol.net

Da der passende Skalierungsmaßstab von der Anwendung abhängt, können Sie durch die Kreuzvalidierung des Vorhersagealgorithmus testen, wie gut ein Set von Maßstäben ist. Bei der Kreuzvalidierung werden Elemente aus der Datenmenge entfernt, und dann wird versucht zu ermitteln, wie gut sie mit dem Rest der Daten vorhergesagt werden können. Abbildung 12-11 zeigt, wie das funktioniert.

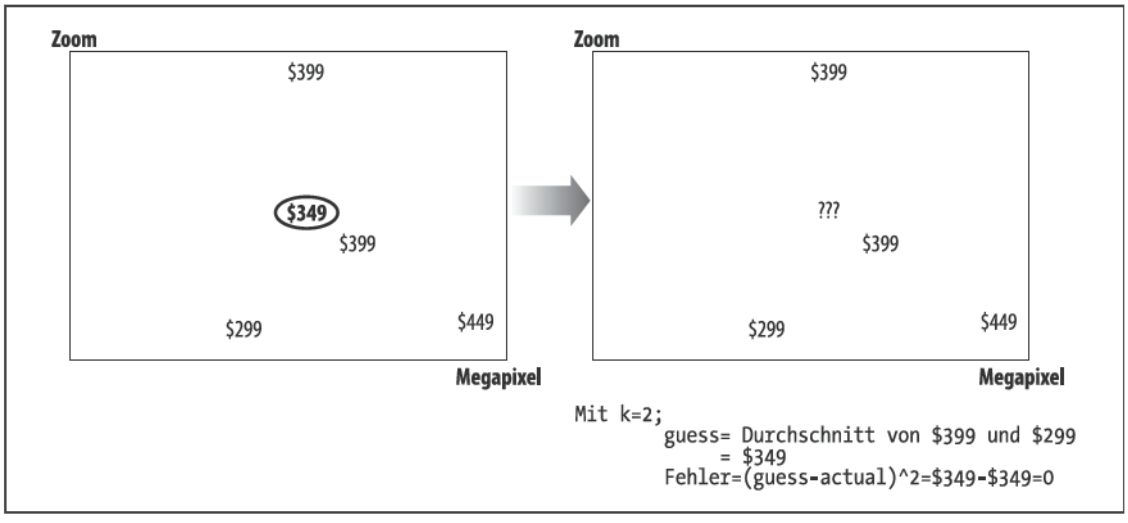

Abbildung 12-11: Kreuzvalidierung eines einzelnen Elements

Indem Sie die Kreuzvalidierung mit vielen verschiedenen Skalierungsfaktoren anwenden, können Sie für jedes Element eine Fehlerrate ermitteln und diese nutzen, um zu bestimmen, welche Skalierungsfaktoren für die Vorhersage neuer Elemente verwendet werden sollten.

Anwendung Ihres kNN-Codes

In Kapitel 8 haben Sie Funktionen für kNN und gewichtete kNN erstellt. Sie lassen sich leicht auf die Beispiel-Datenmenge aus Tabelle 12-6 anwenden.

```
>>> cameras=[{'input':(7.1,3.8),'result':399},
... {'input':(5.0,2.4),'result':299},
... {'input':(6.0,4.0),'result':349},
... {'input':(6.0,12.0),'result':399},
... {'input':(10.0,3.0),'result':449}]
>>> import numpredict
>>> numpredict(cameras,(6.0,6.0),k=2)
374.0
>>> numpredict.weightedknn(cameras,(6.0,6.0),k=3)
351.52666892719458
```

Es ist möglich, diese Ergebnisse durch Skalieren der Daten zu verbessern. Dazu dient die Funktion rescale:

```
>>> scc=numpredict.rescale(cameras,(1,2))
>>> scc
```

e-bol.net

```
[{'input': [7.1, 7.6], 'result': 399}, {'input': [5.0, 4.8], 'result': 299},
{'input': [6.0, 8.0], 'result': 349}, {'input': [6.0, 24.0], 'result': 399},
{'input': [10.0, 6.0], 'result': 449}]
```

Und durch die Anwendung von `crossvalidate` können Sie ermitteln, welche Skalierungsfaktoren am besten sind:

```
>>> numpredict.crossvalidate(knn1,cameras,test=0.3,trials=2)
3750.0
>>> numpredict.crossvalidate(knn1,scc,test=0.3,trials=2)
2500.0
```

Bei einer Datenmenge mit mehr Variablen wird es sehr mühselig, die möglichen Skalierungsfaktoren zu bestimmen, daher können Sie entweder alle möglichen Werte durchlaufen, um das beste Ergebnis zu bestimmen, oder Sie nutzen, wie in Kapitel 8 gezeigt, einen der Optimierungsalgorithmen.

Stärken und Schwächen

k-nächste Nachbarn ist einer der wenigen Algorithmen, die numerische Vorhersagen in komplexen Funktionen treffen können und dabei trotzdem leicht zu interpretieren sind. Der Begründungsprozess lässt sich gut verstehen, und eine einfache Änderung des Codes ermöglicht Ihnen, genau festzustellen, welche Nachbarn in der Berechnung verwendet werden. Neuronale Netze können ebenfalls numerische Vorhersagen in komplexen Funktionen treffen, aber sie werden mit Sicherheit keine so einfachen Beispiele liefern können, um den Begründungsprozess zu verstehen.

Zudem verbessert das Bestimmen der korrekten Skalierungsfaktoren nicht nur die Vorhersagen, sondern macht Ihnen auch noch deutlich, welche Variablen bei der Vorhersage wichtig sind. Jede Variable, die auf 0 herunterskaliert wurde, kann herausgeschmissen werden. In manchen Fällen waren genau diese Daten nur mit viel Aufwand zu ermitteln, daher kann man in Zukunft eventuell Geld und Zeit sparen, weil man weiß, dass diese Variablen nicht benötigt werden.

kNN ist eine *Online-Technik*, das heißt, neue Daten können jederzeit hinzugefügt werden. Das unterscheidet sie von anderen Techniken wie den Support-Vektor-Maschinen, die beim Ändern von Daten ein neues Training benötigen. Zudem braucht man bei neuen Daten keine neue Berechnung, sie werden einfach dem Set hinzugefügt.

Der größte Nachteil des kNN ist die Notwendigkeit, alle Trainingsdaten vorrätig halten zu müssen, um die Vorhersagen treffen zu können. In einer Datenmenge mit Millionen von Beispielen ist das nicht nur ein Platz-, sondern auch ein Zeitproblem – jedes Element, für das Sie eine Vorhersage treffen wollen, muss mit jedem anderen Element verglichen werden, um zu bestimmen, welche die nahesten sind. Dieser Prozess kann für manche Anwendungen zu lange dauern.

e-bol.net

Ein weiterer Nachteil ist, dass das Ermitteln der korrekten Skalierungsfaktoren sehr mühselig sein kann. Auch wenn es Wege gibt, diesen Prozess mehr zu automatisieren, kann die Kreuzvalidierung und das Bewerten Tausender möglicher Skalierungsfaktoren bei einer großen Datenmenge sehr rechenaufwendig werden. Wenn es viele verschiedene Variablen gibt, die auszuprobieren sind, kann es notwendig sein, Millionen von Skalierungsfaktoren zu testen, bevor der richtige gefunden wird.

Clusteranalyse

Die hierarchische und die K-Means-Clusteranalyse sind beides Techniken zum unüberwachten Lernen, sie erfordern also keine Trainingsdaten, da sie nicht versuchen, Vorhersagen zu treffen. Kapitel 3 hat gezeigt, wie man eine Liste mit Top-Blogs nehmen und sie automatisch so in Gruppen unterteilen kann, dass Sie sehen, welche zusammengehören, weil sie zum Beispiel die gleichen Themen behandeln oder gleiche Wörter verwenden.

Hierarchische Clusteranalyse

Die Clusteranalyse funktioniert mit jeder Sammlung von Elementen, die eine oder mehrere numerische Eigenschaften haben. Das Beispiel in Kapitel 3 hat die Wortfrequenzen der verschiedenen Blogs genutzt, aber man kann ein beliebiges Set von Zahlen zur Clusteranalyse verwenden. Um zu zeigen, wie der hierarchische Clusteralgorithmus funktioniert, nehmen wir eine einfache Tabelle mit Elementen (ein paar Buchstaben des Alphabets) mit numerischen Eigenschaften:

Tabelle 12-7: Einfache Tabelle für die Clusteranalyse

Element	E1	E2
A	1	8
B	3	8
C	2	6
D	1,5	1
E	4	2

Abbildung 12-12 zeigt, wie diese Elemente in Clustern angeordnet werden. Im ersten Bereich wurden die Elemente in zwei Dimensionen aufgetragen, wobei E1 auf der x-Achse und E2 auf der y-Achse liegen. Die hierarchische Clusteranalyse findet nun immer die beiden am nächsten beieinanderliegenden Elemente und verbindet sie zu einem Cluster. Im zweiten Bereich können Sie sehen, dass die beiden am engsten zusammenliegenden Elemente A und B zusammen gruppiert wurden. Die »Position« dieses Clusters ist der Mittelwert der beiden Elemente in ihm. Dieser

e-bol.net

Prozess setzt sich fort, bis im letzten Bereich alles in einem großen Cluster zusammengefasst ist.

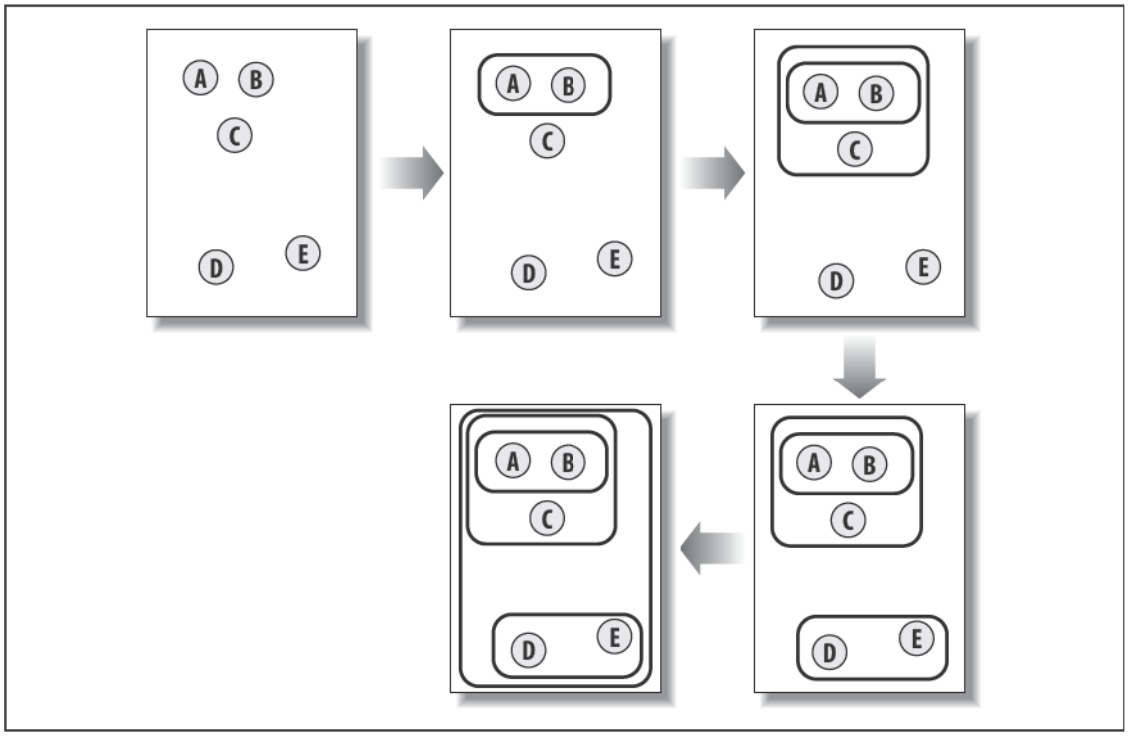

Abbildung 12-12: Vorgehen bei der hierarchischen Clusteranalyse

Dieser Prozess führt zu einer Hierarchie, die als *Dendrogramm* dargestellt werden kann – eine baumartige Struktur, die zeigt, welche Elemente und Gruppen am nächsten beieinanderliegen. Das Dendrogramm für die Beispiel-Datenmenge ist in Abbildung 12-13 zu sehen.

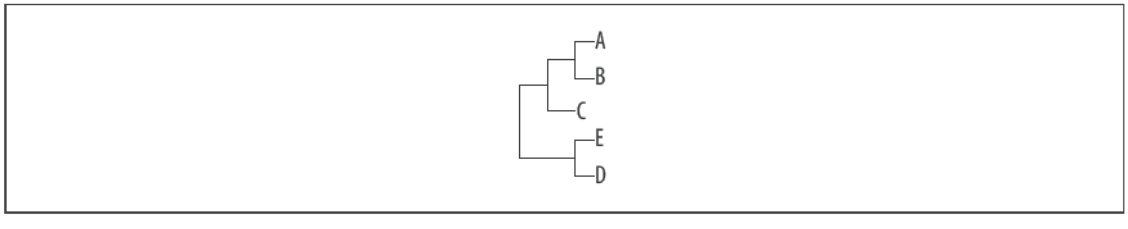

Abbildung 12-13: Dendrogramm mit den gruppierten Buchstaben

Die zwei engsten Elemente A und B sind am Ende zusammengefasst. C ist mit der Kombination aus A und B verbunden. Aus dem Dendrogramm können Sie sich einen beliebigen Verzweigungspunkt herauspicken und entscheiden, ob es sich dabei um eine interessante Gruppe handelt. In Kapitel 3 haben Sie Zweige gesehen, die fast vollständig aus politischen Blogs bestehen, während andere Technologie-Blogs enthalten und so weiter.

e-bol.net

K-Means-Clusteranalyse

Eine andere Methode zum Clustern der Daten wird als K-Means-Clusteranalyse bezeichnet. Während bei der hierarchischen Clusteranalyse ein Baum aus Elementen aufgebaut wird, teilt die K-Means-Clusteranalyse die Daten direkt in unterschiedliche Gruppen ein. Sie müssen dabei im Voraus entscheiden, wie viele Gruppen Sie haben wollen, bevor der Algorithmus gestartet wird. Abbildung 12-14 zeigt ein Beispiel für die K-Means-Clusteranalyse, wobei versucht wird, zwei Cluster in einer Datenmenge zu finden, die von dem bei der hierarchischen Analyse verwendeten leicht abweicht.

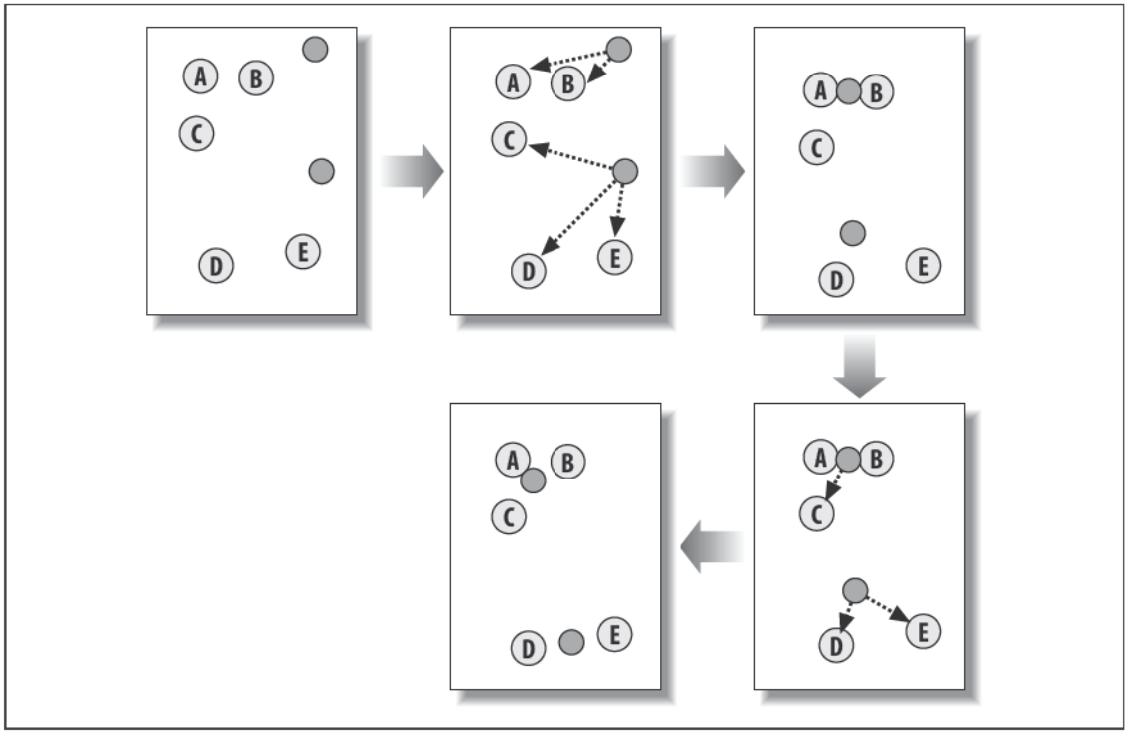

Abbildung 12-14: Ablauf der K-Means-Clusteranalyse

Im ersten Bereich werden die beiden Clusterzentren (als dunklen Kreise dargestellt) zufällig platziert. Bereich 2 zeigt, dass jedes Element dem nächststehenden Clusterzentrum zugewiesen wird – in diesem Fall A und B dem oberen, C, D und E dem unteren Clusterzentrum. Im dritten Bereich wurden die Clusterzentren zur Durchschnittsposition der Elemente verschoben, die ihnen zugeordnet sind. Wenn die Zuweisungen erneut durchgeführt werden, stellt sich heraus, dass C jetzt näher am oberen Clusterzentrum liegt, während D und E beim unteren verbleiben. Im Endergebnis liegen daher A, B und C im oberen Cluster, während sich D und E im unteren befinden.

e-bol.net

Anwendung Ihres Codes für die Clusteranalyse

Um eine Clusteranalyse durchführen zu können, brauchen Sie eine Datenmenge und eine Abstandsmetrik. Die Datenmenge besteht aus Listen mit Zahlen, wobei jede Zahl eine Variable repräsentiert. In Kapitel 3 wurden sowohl die Pearson-Korrelation als auch der Tanimoto-Score als Abstandsmetriken verwendet, aber man kann auch leicht auf andere zurückgreifen, wie zum Beispiel die euklidische Distanz:

```
>>> data=[[1.0,8.0],[3.0,8.0],[2.0,7.0],[1.5,1.0],[4.0,2.0]]
>>> labels=['A','B','C','D','E']
>>> def euclidean(v1,v2): return sum([(v1[i]-v2[i])**2 for i in range(len(v1))])
>>> import clusters
>>> hcl=clusters.hcluster(data,distance=euclidean)
>>> kcl=clusters.kcluster(data,distance=euclidean,k=2)
Iteration 0
Iteration 1
```

Bei der K-Means-Clusteranalyse können Sie die Ergebnisse für jedes Ihrer zwei Cluster einfach ausgeben:

```
>>> kcl
[[0, 1, 2], [3, 4]]
>>> for c in kcl: print [labels[l] for l in c]
...
['A', 'B', 'C']
['D', 'E']
```

Die Ergebnisse der hierarchischen Clusteranalyse lassen sich nicht so schön als Text ausgeben, aber der in Kapitel 3 erstellte Code enthält auch eine Funktion, mit der das Dendrogramm gezeichnet werden kann:

```
>>> clusters.drawdendrogram(hcl,labels,jpeg='hcl.jpg')
```

Die Wahl des passenden Algorithmus hängt stark davon ab, was Sie tun wollen. Es ist häufig nützlich, die Daten in separate Gruppen aufzuteilen, wie Sie es bei der K-Means-Clusteranalyse machen. Denn dabei ist es viel einfacher, die Gruppen auszugeben und zu charakterisieren. Andererseits kann es bei einer vollständig neuen Datenmenge sein, dass Sie im Voraus nicht wissen, wie viele Gruppen Sie benötigen und Sie zudem herausfinden wollen, welche Gruppen untereinander am ähnlichsten sind. In diesem Fall wäre die hierarchische Clusteranalyse die bessere Wahl.

Es ist auch möglich, die Vorteile beider Methoden zu kombinieren, indem man erst die K-Means-Analyse nutzt, um eine Reihe von Gruppen zu erstellen, und diese Gruppen dann hierarchisch clustert, wobei die Abstände zwischen den Clusterzentren genutzt werden. Damit erhalten Sie separate Gruppen, die aber in einem Baum angeordnet sind, sodass Sie erkennen können, wie die Gruppen zueinander in Beziehung stehen.

e-bol.net

Multidimensionales Skalieren

Eine andere in Kapitel 3 behandelte Methode, die ebenfalls auf die Blogs angewendet wurde, ist das multidimensionale Skalieren. Wie bei der Clusteranalyse handelt es sich dabei um eine Technik des unüberwachten Lernens, die nicht versucht, Vorhersagen zu treffen, sondern aufzeigen will, welche Elemente zusammenpassen. Dabei wird eine niedrigdimensionalere Repräsentation einer Datenmenge erstellt, bei der die Abstände so nahe wie möglich an der ursprünglichen Datenmenge liegen. Zur Anzeige auf einem Monitor oder auf Papier bedeutet dies normalerweise ein Skalieren der Daten von vielen Dimensionen auf zwei Dimensionen.

Angenommen, Sie hätten die in Tabelle 12-8 gezeigte vierdimensionale Datenmenge (jedes Element hat vier zugewiesene Werte).

Tabelle 12-8: Einfache vierdimensionale Tabelle zum Skalieren

A	0,5	0,0	0,3	0,1
B	0,4	0,15	0,2	0,1
C	0,2	0,4	0,7	0,8
D	1,0	0,3	0,6	0,0

Mithilfe der euklidischen Abstandsformel können Sie einen Distanzwert für jedes Elementpaar ermitteln. So ist der Abstand zwischen A und B zum Beispiel *sqrt(0.1^2+0.15^2+0.1^2+0.0^2)* = 0,2. Die vollständige Matrix aller paarweisen Abstände finden Sie in Tabelle 12-9.

Tabelle 12-9: Abstands-Matrix für das Beispiel

	A	B	C	D
A	0,0	0,2	0,9	0,8
B	0,2	0,0	0,9	0,7
C	0,9	0,9	0,0	1,1
D	0,8	0,7	1,1	0,0

Das Ziel ist hierbei, alle Elemente in einem zweidimensionalen Graphen so zu zeichnen, dass die Abstände in zwei Dimensionen so nahe wie möglich an den Abständen in vier Dimensionen sind. Alle Elemente werden zufällig auf dem Graphen verteilt, und die aktuellen Abstände zwischen ihnen werden berechnet (siehe Abbildung 12-15).

Für jedes Elementpaar wird der gewünschte Abstand mit dem aktuellen Abstand verglichen und ein Fehlerwert ermittelt. Jedes Element wird nun um einen kleinen Betrag im Verhältnis des Fehlerwerts verschoben. Abbildung 12-16 zeigt die Kräfte, die auf Element A wirken. Der Abstand zwischen A und B beträgt im Graphen 0,5,

e-bol.net

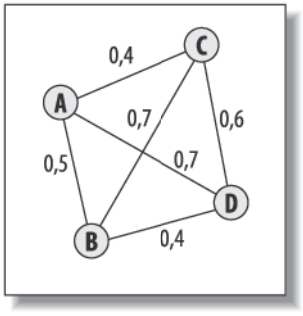

Abbildung 12-15: Abstände zwischen Elementen

aber der gewünschte Abstand ist nur 0,2, daher muss A näher an B gebracht werden. Gleichzeitig wird A auch von C und D weggeschoben, da es zu nahe an ihnen liegt.

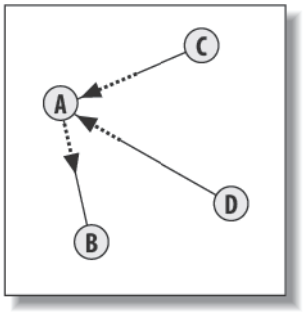

Abbildung 12-16: Kräfte, die auf Element A wirken

Jeder Knoten wird entsprechend der Kombination aller anderen Knoten, die auf ihn wirken, verschoben. Bei jeder Runde sollte der Unterschied zwischen den aktuellen Abständen und den Zielabständen ein bisschen kleiner werden. Diese Prozedur wird so oft wiederholt, bis der Gesamtfehler durch das Verschieben von Elementen nicht mehr reduziert werden kann.

Anwendung Ihres Codes zur multidimensionalen Skalierung

In Kapitel 3 haben Sie zwei Funktionen zur multidimensionalen Skalierung erstellt – eine zum Ausführen des Algorithmus und die andere zum Anzeigen der Ergebnisse. Die erste Funktion `scaledown` erwartet die Liste mit den Elementwerten in den vielen Dimensionen und liefert die Liste in der gleichen Reihenfolge zurück, aber reduziert auf zwei Dimensionen:

```
>>> labels=['A','B','C','D']
>>> scaleset=[[0.5,0.0,0.3,0.1],
... [0.4,0.15,0.2,0.1],
... [0.2,0.4,0.7,0.8],
```

e-bol.net

```
... [1.0,0.3,0.6,0.0]]
>>> twod=clusters.scaledown(scaleset,distance=euclidean)
>>> twod
[[0.45, 0.54],
 [0.40, 0.54],
 [-0.30, 1.02],
 [0.92, 0.59]]
```

Die zweite Funktion `draw2d` nimmt nun diese herunterskalierte Liste und erstellt ein Bild:

```
>>> clusters.draw2d(twod,labels,jpeg='abcd.jpg')
```

Damit wird eine Datei namens *abcd.jpg* erstellt, die das Ergebnis enthält. Sie können das Ergebnis auch anders darstellen, indem Sie die von `scaledown` erzeugte Liste nehmen und in einem anderen Programm verwenden, zum Beispiel in einer Tabellenkalkulation.

Nicht-negative Matrix-Faktorisierung

Kapitel 10 behandelte eine fortgeschrittenere Technik namens nicht-negative Matrix-Faktorisierung (NMF), mit deren Hilfe man eine Reihe von numerischen Beobachtungen in ihre Komponenten aufteilen kann. Diese Methode wurde dazu genutzt zu zeigen, wie Nachrichten aus verschiedenen Themen zusammengesetzt werden können und wie sich das Handelsvolumen verschiedener Aktien in Ereignisse aufteilen lässt, die die einzelnen Aktien oder mehrere Aktien gleichzeitig beeinflussen. Dies ist ebenfalls ein Algorithmus zum unüberwachten Lernen, da er dabei hilft, Daten zu charakterisieren, anstatt Vorhersagen über Kategorien oder Werte zu treffen.

Um zu verstehen, was die NMF tut, schauen Sie sich die Werte in Tabelle 12-10 an:

Tabelle 12-10: Einfache Tabelle für die NMF

Beobachtungsnummer	A	B
1	29	29
2	43	33
3	15	25
4	40	28
5	24	11
6	29	29
7	37	23
8	21	6

Sie gehen davon aus, dass die Beobachtungen A und B irgendwie aus zwei Zahlenpaaren (den Merkmalen) kombiniert sind, aber Sie wissen nicht, was diese Paare

e-bol.net

sind oder wie viel jedes Paar zu jeder Beobachtung beiträgt (die Gewichtung). Die NMF kann mögliche Werte sowohl für die Merkmale als auch für die Gewichtungen finden. Als Sie in Kapitel 10 mit den Nachrichten gearbeitet haben, handelte es sich bei den Beobachtungen um die Artikel und bei den Spalten um die Wörter in den Artikeln. Bei den Aktienhandelsvolumen waren die Beobachtungen die Tage und die Spalten die verschiedenen Aktienticker. In beiden Fällen versuchte der Algorithmus, eine kleinere Anzahl von Elementen zu finden, die man in unterschiedlicher Gewichtung kombinieren kann, um die verschiedenen Beobachtungen zu erhalten.

Eine mögliche Antwort für die Daten in der Tabelle sind die beiden Paare (3; 5) und (7; 2).

Damit können Sie die Beobachtungen wieder zusammenbauen, indem Sie die Paare in unterschiedlicher Gewichtung kombinieren, zum Beispiel so:

$$5^*(3; 5) + 2^*(7; 2) = (29; 29)$$
$$5^*(3; 5) + 4^*(7; 2) = (43; 33)$$

Dies lässt sich auch als Matrix-Multiplikation darstellen, wie in Abbildung 12-17 zu sehen:

$$\underbrace{\begin{pmatrix} 5 & 2 \\ 5 & 4 \\ 5 & 0 \\ 4 & 4 \\ 1 & 3 \\ 5 & 2 \\ 3 & 4 \\ 0 & 3 \end{pmatrix}}_{\text{Gewichtung}} \mathbf{X} \underbrace{\begin{pmatrix} 3 & 5 \\ 7 & 2 \end{pmatrix}}_{\text{Merkmale}} = \underbrace{\begin{pmatrix} 29 & 29 \\ 43 & 33 \\ 15 & 25 \\ 40 & 28 \\ 24 & 11 \\ 29 & 29 \\ 37 & 23 \\ 21 & 6 \end{pmatrix}}_{\text{Datenmenge}}$$

Abbildung 12-17: Faktorisieren einer Datenmenge in Gewichtungen und Merkmale

Das Ziel der NMF ist ein automatisches Finden der Merkmals-Matrix und der Gewichtungs-Matrix. Dazu beginnt sie in beiden Fällen mit Zufalls-Matrizen und passt sie entsprechend der *Update Rules* an. Die Regeln erzeugen vier neue Matrizen. In den folgenden Beschreibungen wird die Ursprungs-Matrix als Daten-Matrix bezeichnet:

hn
: Die transponierte Gewichtungs-Matrix, multipliziert mit der Daten-Matrix.

hd
: Die transponierte Gewichtungs-Matrix, multipliziert mit der Gewichtungs-Matrix und der Merkmals-Matrix.

wn
: Die Daten-Matrix multipliziert mit der transponierten Merkmals-Matrix.

e-bol.net

wd

Die Gewichtungs-Matrix, multipliziert mit der Merkmals-Matrix und der transponierten Gewichtungs-Matrix.

Um die Merkmals- und die Gewichtungs-Matrizen zu aktualisieren, werden alle Matrizen in Arrays umgewandelt. Jeder Wert in der Merkmals-Matrix wird mit dem entsprechenden Wert in *hn* multipliziert und durch den entsprechenden Wert in *hd* dividiert. Genauso wird jeder Wert in der Gewichtungs-Matrix mit dem Wert in *wn* multipliziert und durch den Wert in *wd* dividiert.

Diese Schritte werden wiederholt, bis das Produkt aus Merkmals- und Gewichtungs-Matrix nahe genug an der Daten-Matrix ist. Die Merkmals-Matrix kann Ihnen die zugrunde liegenden Ursachen nennen, aus denen Sie Ihre Datenmenge zusammensetzen können, wie zum Beispiel Nachrichten oder Ereignisse im Aktienmarkt.

Anwendung Ihres NMF-Codes

Um den NMF-Code zu nutzen, müssen Sie nur die Funktion `factorize` mit einer Liste mit Beobachtungen und der Anzahl der Merkmale aufrufen, die Sie finden wollen:

```
>>> from numpy import *
>>> import nmf
>>> data=matrix([[ 29.,  29.],
... [ 43.,  33.],
... [ 15.,  25.],
... [ 40.,  28.],
... [ 24.,  11.],
... [ 29.,  29.],
... [ 37.,  23.],
... [ 21.,   6.]])
>>> weights,features=nmf.factorize(data,pc=2)
>>> weights
matrix([[ 0.64897525,  0.75470755],
        [ 0.98192453,  0.80792914],
        [ 0.31602596,  0.70148596],
        [ 0.91871934,  0.66763194],
        [ 0.56262912,  0.22012957],
        [ 0.64897525,  0.75470755],
        [ 0.85551414,  0.52733475],
        [ 0.49942392,  0.07983238]])
>>> features
matrix([[ 41.62815416,   6.80725866],
        [  2.62930778,  32.57189835]])
```

Die Gewichtungen und die Merkmale werden zurückgeliefert. Sie werden nicht jedes Mal gleich sein, da es durchaus mehrere gültige Merkmalssets bei einem kleinen Beobachtungsset geben kann. Je größer das Beobachtungsset ist, desto größer

e-bol.net

ist die Wahrscheinlichkeit, dass die Ergebnisse konsistent sein werden, auch wenn die Merkmale eventuell in anderer Reihenfolge vorkommen.

Optimierung

Die in Kapitel 5 behandelte Optimierung unterscheidet sich ein wenig von den anderen Methoden – anstatt mit einer Datenmenge zu arbeiten, versucht sie, Werte auszuwählen, die das Ergebnis einer Zielfunktion minimieren. Kapitel 5 hat mehrere Beispiele für Zielfunktionen präsentiert, wie zum Beispiel für die Planung einer Gruppenreise, bei der die Kombination aus Preis und Wartezeit am Flughafen genutzt wird, das Zuweisen von Wohnheimzimmern an Studenten und das Optimieren des Layouts in einem einfachen Graph. Nachdem die Zielfunktion einmal entworfen wurde, können die gleichen Algorithmen genutzt werden, um diese drei völlig verschiedenen Probleme zu lösen. Zwei Algorithmen wurden behandelt: die simulierte Abkühlung und genetische Algorithmen.

Die Zielfunktion

Eine Zielfunktion ist eine beliebige Funktion, die eine Schätzung einer Lösung erwartet und einen Wert zurückliefert, der für schlechtere Lösungen größer und für bessere Lösungen kleiner ist. Optimierungsalgorithmen verwenden diese Funktion, um Lösungen zu bewerten und die möglichen Lösungen nach der besten zu durchsuchen. Die Zielfunktion, die Sie bei der Optimierung nutzen, muss meist viele Variablen berücksichtigen, und es ist nicht immer offensichtlich, welche Variable geändert werden sollte, um das Ergebnis zu verbessern. Trotzdem wollen wir uns zur Veranschaulichung eine Funktion mit nur einer Variablen anschauen:

```
y = 1/x * sin(x)
```

Abbildung 12-18 zeigt den Graphen den Funktion.

Da die Funktion nur eine Variable hat, kann man im Graphen leicht erkennen, wo der niedrigste Punkt liegt. Wir nutzen sie, damit Sie sehen können, wie die Optimierung funktioniert – bei echten Problemen werden Sie nicht dazu in der Lage sein, kompliziertere Funktionen mit vielen Variablen grafisch darzustellen, um den niedrigsten Punkt zu finden.

Besonders interessant an dieser Funktion ist, dass sie viele lokale Minima hat. Dabei handelt es sich um Punkte, die niedriger als die umgebenden Punkte, aber nicht notwendigerweise insgesamt am niedrigsten sind. Das bedeutet, dass das Problem nicht einfach dadurch gelöst werden kann, dass man einen zufälligen Punkt auswählt und sich dann den »Hang« nach unten bewegt, da man dann in einem lokalen Minimum hängen bleiben kann und das globale Minimum niemals findet.

e-bol.net

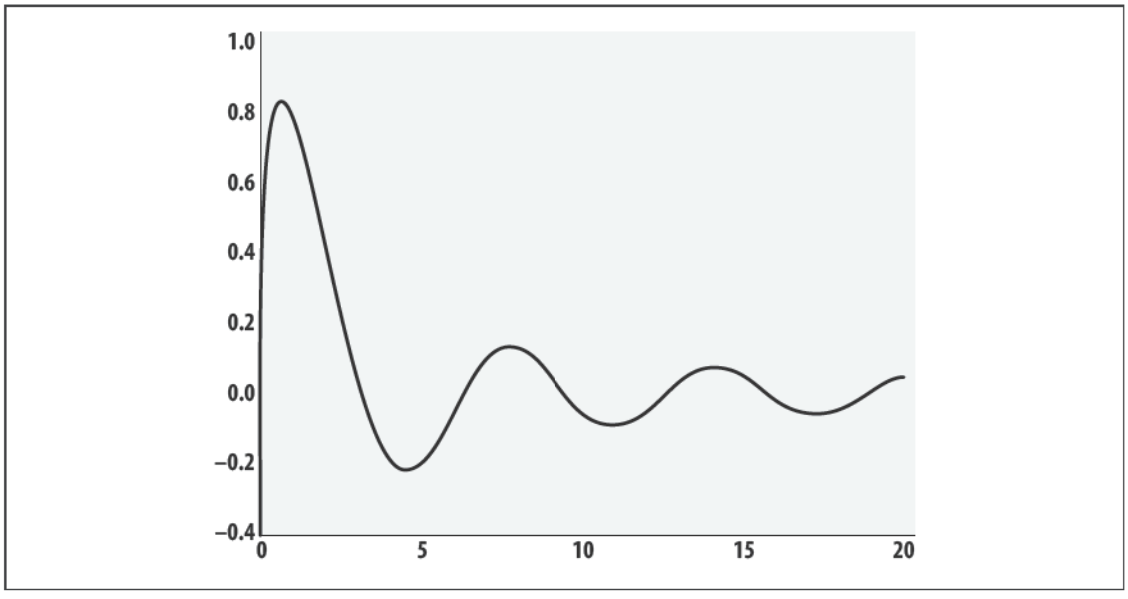

*Abbildung 12-18: Graph der Funktion 1/x * sin x*

Simulierte Abkühlung

Die simulierte Abkühlung, angelehnt an das Ausglühen von Legierungen in der Physik, geht von einer zufällig gewählten Lösung aus. Sie versucht, die Lösung zu verbessern, indem sie die Kosten einer ähnlichen Lösung bestimmt, die sich einen kleinen Schritt entfernt in zufällige Richtung befindet. Wenn die Kosten kleiner sind, wird dies die neue Lösung. Sind sie höher, wird dies trotzdem mit einer gewissen Wahrscheinlichkeit die neue Lösung. Diese Wahrscheinlichkeit hängt von der aktuellen *Temperatur* ab. Diese Temperatur beginnt mit einem hohen Wert und sinkt langsam ab, sodass der Algorithmus zu Beginn noch sehr wahrscheinlich schlechtere Lösungen akzeptiert, um zu vermeiden, dass man in einem lokalen Minimum hängen bleibt.

Wenn die Temperatur den Wert 0 erreicht hat, liefert der Algorithmus die aktuelle Lösung zurück.

Genetische Algorithmen

Genetische Algorithmen wurden durch die Evolutionstheorie inspiriert. Ein genetischer Algorithmus beginnt mit vielen zufällig gewählten Lösungen, die als Population bezeichnet werden. Die stärksten Mitglieder der Population – die mit den geringsten Kosten – werden ausgewählt und durch kleine Änderungen (Mutation) oder die Kombination von Eigenschaften (Crossover beziehungsweise Züchten) angepasst. Damit wird eine neue Population erzeugt, die als nächste Generation bezeichnet wird, wobei sich mit weiteren Generationen die Lösungen verbessert.

e-bol.net

Der Prozess endet, wenn ein bestimmter Schwellenwert erreicht wird, die Population sich über mehrere Generationen nicht verbessert hat oder eine maximale Zahl an Generationen erreicht wurde. Der Algorithmus liefert die beste Lösung zurück, die in einer der Generationen gefunden wurde.

Anwendung Ihres Optimierungscodes

Bei beiden Algorithmen müssen Sie eine Zielfunktion definieren und die Domäne der Lösung bestimmen. Die Domäne ist der mögliche Bereich jeder Variablen. In diesem einfachen Beispiel können Sie `[(0,20)]` nutzen, was bedeutet, dass es eine Variable gibt, die zwischen 0 und 20 liegen kann. Beide Optimierungsmethoden können dann mit der Funktion `costf` aufgerufen werden, wobei die `domain` als weiterer Parameter übergeben wird:

```
>>> import math
>>> def costf(x): return (1.0/(x[0]+0.1))*math.sin(x[0])
>>> domain=[(0,20)]
>>> optimization.annealingoptimize(domain,costf)
[5]
```

Bei fast jedem Problem wird es notwendig sein, die Optimierung ein paarmal laufen zu lassen, um die Parameter anzupassen und die richtige Balance zwischen der Laufzeit und der Qualität der Lösung zu finden. Wenn ein Optimierer für eine Reihe ähnlicher Probleme erstellt wird – zum Beispiel Reiseplanungen, bei denen das Ziel gleich bleibt, sich aber die zugrunde liegenden Details ändern (Flugzeiten und Preise) –, können Sie mit den Parametern einmal experimentieren, sich dann für funktionierende Einstellungen entscheiden und sie schließlich beibehalten.

Es gibt viele Möglichkeiten, maschinelles Lernen, offene APIs und die Beiträge einer Gemeinschaft zu kombinieren, und es wird in Zukunft auch noch mehr geben. Denn die Algorithmen werden verfeinert, immer mehr APIs öffnen sich, und immer mehr Menschen nehmen aktiv im Netz teilt. Ich hoffe, dass dieses Buch Ihnen die Werkzeuge und die Inspiration für viele neue Herausforderungen geliefert hat!

e-bol.net

ANHANG A

Fremdbibliotheken

Dieses Buch hat eine Reihe von Bibliotheken von dritter Seite vorgestellt, die wir zum Einsammeln, Speichern und Analysieren von Daten verwendet haben. Dieser Anhang behandelt das Herunterladen und die Installation dieser Bibliotheken. Zusätzlich werden auch noch ein paar Beispiele zur Verwendung aufgeführt.

Universal Feed Parser

Der Universal Feed Parser ist eine Python-Bibliothek, die von Mark Pilgrim zum Parsen von RSS- und Atom-Feeds geschrieben wurde. Diese Bibliothek wird im ganzen Buch verwendet, um Blog-Posts und Artikel von Nachrichtenseiten einfach herunterladen zu können. Die Homepage der Bibliothek ist *http://feedparser.org*.

Installation für alle Plattformen

Die Seite zum Herunterladen der Bibliothek ist *http://code.google.com/p/feedparser/downloads/list*. Laden Sie die neueste Version der Datei mit dem Namen *feedparser-X.Y.zip*.

Extrahieren Sie den Inhalt der ZIP-Datei in ein leeres Verzeichnis. An der Befehlszeile geben Sie Folgendes ein:

```
c:\download\feedparser>python setup.py install
```

Damit wird Ihre Python-Installation gesucht und die Bibliothek dort installiert. Nach der Installation können Sie am Python-Prompt **`import feedparser`** eingeben, um sie zu nutzen.

Beispiele für die Anwendung der Bibliothek finden Sie auf *http://feedparser.org/*.

e-bol.net

Python Imaging Library

Die Python Imaging Library (PIL) ist eine Open Source-Bibliothek, die Python um Fähigkeiten zur Bilderstellung und -verarbeitung ergänzt. Sie unterstützt viele verschiedene Zeichenoperationen und Dateiformate. Die Homepage dieser Bibliothek ist *http://www.pythonware.com/products/pil*.

Installation unter Windows

Die PIL hat einen Windows-Installer, den man herunterladen kann. Auf der Homepage der Bibliothek scrollen Sie zum Download-Abschnitt und laden die neueste Windows-Datei für Ihre Python-Version herunter. Starten Sie sie und folgen Sie den Anweisungen auf dem Bildschirm.

Installation auf anderen Plattformen

Für Nicht-Windows-Plattformen müssen Sie die Bibliothek aus den Quellen aufbauen. Diese finden sich zum Herunterladen auf der Homepage der Bibliothek und funktionieren mit jeder neueren Python-Version.

Zur Installation laden Sie die Quellen der neuesten Version herunter und geben an der Befehlszeile die folgenden Kommandos ein, wobei Sie 1.1.6 durch die Version ersetzen, die Sie heruntergeladen haben:

```
$ gunzip Imaging-1.1.6.tar.gz
$ tar xvf Imaging-1.1.6.tar
$ cd Imaging-1.1.6
$ python setup.py install
```

Damit werden die Erweiterungen kompiliert und die Bibliothek in Ihrem Python-Verzeichnis installiert.

Einfaches Anwendungsbeispiel

Dieses Beispiel erstellt ein kleines Bild, zeichnet ein paar Linien und gibt Text aus. Dann speichert es das Bild als JPEG-Datei.

```
>>> from PIL import Image,ImageDraw
>>> img=Image.new('RGB',(200,200),(255,255,255))  # 200 x 200, weißer Hintergrund
>>> draw=ImageDraw.Draw(img)
>>> draw.line((20,50,150,80),fill=(255,0,0))      # rote Linie
>>> draw.line((150,150,20,200),fill=(0,255,0))    # grüne Linie
>>> draw.text((40,80),'Hello!',(0,0,0))           # schwarzer Text
>>> img.save('test.jpg','JPEG')                   # speichern in test.jpg
```

Eine umfangreichere Sammlung an Beispielen finden Sie unter *http://www.pythonware.com/library/pil/handbook/introduction.htm*.

e-bol.net

Beautiful Soup

Beautiful Soup ist ein Python-Parser für HTML- und XML-Dokumente. Er ist darauf ausgelegt, auch schlecht geschriebene Webseiten zu parsen. Der Parser wird in diesem Buch genutzt, um Datenmengen aus Websites zu erstellen, die keine API bieten, und um alle Texte auf einer Seite für eine Indexierung zu finden. Die Homepage für diese Bibliothek ist *http://www.crummy.com/software/BeautifulSoup*.

Installation auf allen Plattformen

Beautiful Soup ist als einzelne Quelldatei herunterzuladen. Ziemlich am Ende der Homepage finden Sie einen Link, um *BeautifulSoup.py* zu laden. Tun Sie das und stecken Sie die Datei entweder in Ihr Arbeitsverzeichnis oder in *Python/Lib*.

Einfaches Anwendungsbeispiel

Dieses Beispiel parst den HTML-Code der Google-Homepage und zeigt, wie man Elemente aus dem DOM extrahiert sowie nach Links sucht.

```
>>> from BeautifulSoup import BeautifulSoup
>>> from urllib import urlopen
>>> soup=BeautifulSoup(urlopen('http://google.com'))
>>> soup.head.title
<title>Google</title>
>>> links=soup('a')
>>> len(links)
21
>>> links[0]
<a href="http://www.google.com/ig?hl=en">iGoogle</a>
>>> links[0].contents[0]
u'iGoogle'
```

Eine Reihe ausführlicherer Beispiele finden Sie unter *http://www.crummy.com/software/BeautifulSoup/documentation.html*.

pysqlite

pysqlite ist eine Python-Schnittstelle zur eingebetteten Datenbank SQLite. Anders als klassische Datenbanken läuft eine eingebettete Datenbank nicht auf eigenen Serverprozessen, daher lässt sie sich viel leichter installieren und einrichten. SQLite speichert zudem die gesamte Datenbank in einer einzelnen Datei. Dieses Buch nutzt pysqlite, um zu zeigen, wie man eingesammelte Daten persistieren kann.

Die Homepage für pysqlite ist *http://www.initd.org/tracker/pysqlite/wiki/pysqlite*.

e-bol.net

Installation unter Windows

Die Hompage enthält Links zum Herunterladen eines Installationsprogramms für Windows. Laden Sie diese Datei und führen Sie sie einfach aus. Sie wird Sie fragen, wo sich Ihre Python-Installation befindet, und sie dort installieren.

Installation auf anderen Plattformen

Für andere Plattformen müssen Sie pysqlite aus den Quellen aufbauen. Diese stehen als Tarball auf der Homepage von pysqlite zur Verfügung. Laden Sie die neueste Version herunter und geben Sie an der Befehlszeile die folgenden Kommandos ein, wobei Sie 2.3.3 durch die von Ihnen heruntergeladene Version ersetzen:

```
$ gunzip pysqlite-2.3.3.tar.gz
$ tar xvf pysqlite-2.3.3.tar.gz
$ cd pysqlite-2.3.3
$ python setup.py build
$ python setup.py install
```

Einfaches Anwendungsbeispiel

Dieses Beispiel erstellt eine neue Tabelle, fügt eine Zeile hinzu und führt ein Commit aus, um die Änderungen zu bestätigen. Dann wird die Tabelle abgefragt:

```
>>> from pysqlite2 import dbapi2 as sqlite
>>> con=sqlite.connect('test1.db')
>>> con.execute('create table people (name,phone,city)')
<pysqlite2.dbapi2.Cursor object at 0x00ABE770>
>>> con.execute('insert into people values ("toby","555-1212","Boston")')
<pysqlite2.dbapi2.Cursor object at 0x00AC8A10>
>>> con.commit()
>>> cur=con.execute('select * from people')
>>> cur.next()
(u'toby', u'555-1212', u'Boston')
```

Beachten Sie, dass Feldtypen in SQLite optional sind. Um diese Anweisungen mit einer klassischeren Datenbank ausführen zu können, müssen Sie wahrscheinlich noch die Feldtypen beim Erstellen der Tabellen zur Deklaration hinzufügen.

NumPy

NumPy ist eine mathematische Bibliothek für Python, die ein Array-Objekt, Funktionen der linearen Algebra und Fourier-Transformationen bereitstellt. Sie ist bei wissenschaftlichen Python-Anwendungen sehr beliebt, und aufgrund seiner wachsenden Popularität ersetzt sie sogar manchmal Tools wie MATLAB. NumPy wird in Kapitel 10 verwendet, um den NMF-Algorithmus zu implementieren. Die Homepage von NumPy ist *http://numpy.scipy.org*.

e-bol.net

Installation unter Windows

Es gibt einen Installer für Windows, der unter *http://sourceforge.net/project/showfiles.php?group_id=1369&package_id=175103* zur Verfügung steht.

Laden Sie die *.exe*-Datei herunter, die zu Ihrer Python-Version passt, und führen Sie sie aus. Sie werden nach dem Verzeichnis gefragt, in dem sich Python befindet, anschließend wird die Bibliothek dort installiert.

Installation auf anderen Plattformen

Auf anderen Plattformen kann NumPy aus den Quellen installiert werden, die Sie ebenfalls unter *http://sourceforge.net/project/showfiles.php?group_id=1369&package_id=175103* herunterladen können.

Laden Sie die *tar.gz*-Datei, die zu Ihrer Python-Version passt. Um die Installation durchzuführen, geben Sie folgende Befehle ein, wobei Sie 1.0.2 durch die von Ihnen heruntergeladene Version ersetzen:

```
$ gunzip numpy-1.0.2.tar.gz
$ tar xvf numpy-1.0.2.tar.gz
$ cd numpy-1.0.2
$ python setup.py install
```

Einfaches Anwendungsbeispiel

Dieses Beispiel zeigt Ihnen, wie Sie Matrizen erstellen, sie miteinander multiplizieren, dann transponieren und schließlich »flachklopfen«:

```
>>> from numpy import *
>>> a=matrix([[1,2,3],[4,5,6]])
>>> b=matrix([[1,2],[3,4],[5,6]])
>>> a*b
matrix([[22, 28],
        [49, 64]])
>>> a.transpose()
matrix([[1, 4],
        [2, 5],
        [3, 6]])
>>> a.flatten()
matrix([[1, 2, 3, 4, 5, 6]])
```

matplotlib

matplotlib ist eine Bibliothek für 2-D-Grafiken in Python, die beim Erstellen mathematischer Graphen deutlich hilfreicher ist als die Python Imaging Library. Die so erzeugten Abbildungen sind aufgrund ihrer guten Qualität dafür gedacht, auch in Publikationen verwendet zu werden.

e-bol.net

Installation

Vor der Installion der matplotlib müssen Sie NumPy installieren, wie dies im vorigen Abschnitt beschrieben wurde. Die matplotlib hat fertige Builds für alle wichtigen Plattformen, unter anderem Windows, Mac OS X, RPM-basierte Linux-Distributionen und Debian-basierte Distributionen. Sie finden detaillierte Anweisungen zur Installation auf einer der Plattformen unter *http://matplotlib.sourceforge.net/installing.html*.

Einfaches Anwendungsbeispiel

Dieses Beispiel wird orangefarbene Kreise nutzen, um vier Punkte an (1;1), (2;4), (3;9) und (4;16) zu zeichnen. Dann wird es die Ausgabe in einer Datei speichern und sie in einem Fenster auf dem Bildschirm anzeigen.

```
>>> from pylab import *
>>> plot([1,2,3,4], [1,4,9,16], 'ro')
[<matplotlib.lines.Line2D instance at 0x01878990>]
>>> savefig('test1.png')
>>> show( )
```

Eine tolle Sammlung von Beispielen finden Sie unter *http://matplotlib.sourceforge.net/tutorial.html*.

pydelicious

pydelicious ist eine Bibliothek, mit der Daten von der Social Bookmarking Site *del.icio*.us geladen werden können. *del.icio.us* stellt eine offizielle API zur Verfügung, die für manche Funktionen genutzt wird, aber pydelicious hält noch ein paar Features mehr bereit, die wir in Kapitel 2 verwendet haben, um die Empfehlungs-Engine aufzubauen. pydelicious wird nun bei Google Code gehostet – Sie finden sie unter *http://code.google.com/p/pydelicious/source*.

Installation für alle Plattformen

Das Herunterladen der neuesten Version von pydelicious ist am einfachsten, wenn Sie die Versionskontrolle *Subversion* installiert haben. In dem Fall können Sie einfach folgenden Befehl eingeben:

```
svn checkout http://pydelicious.googlecode.com/svn/trunk/pydelicious.py
```

Wenn Sie Subversion nicht haben, können die Dateien unter *http://pydelicious.googlecode.com/svn/trunk* heruntergeladen werden.

Nun führen Sie einfach **`python setup.py install`** in dem Verzeichnis aus, in dem die heruntergeladenen Dateien liegen. Damit wird pydelicious in Ihrem Python-Verzeichnis installiert.

e-bol.net

Einfaches Anwendungsbeispiel

pydelicious bietet eine Reihe von Funktionen an, um beliebte Lesezeichen oder die Lesezeichen eines Benutzers herunterzuladen. Zudem ermöglicht sie Ihnen, Ihrem eigenen Konto neue Lesezeichen hinzuzufügen:

```
>> import pydelicious
>> pydelicious.get_popular(tag='programming')
[{'count': '', 'extended': '', 'hash': '',
  'description': u'How To Write Unmaintainable Code',
  'tags': '', 'href': u'http://thc.segfault.net/root/phun/unmaintain.html',
  'user': u'dorsia', 'dt': u'2006-08-19T09:48:56Z'},
 {'count': '', 'extended': '', 'hash': '',
  'description': u'Threading in C#', 'tags': '',
  'href':u'http://www.albahari.com/threading/', etc...
>> pydelicious.get_userposts('dorsia')
[{'count': '', 'extended': '', 'hash': '',
  'description': u'How To Write Unmaintainable Code',
  'tags': '', 'href': u'http://thc.segfault.net/root/phun/unmaintain.html',
  'user': u'dorsia', 'dt': u'2006-08-19T09:48:56Z'}, etc...
>>> a = pydelicious.apiNew(user, passwd)
>>> a.posts_add(url="http://my.com/", desciption="my.com",
                extended="the url is my.moc", tags="my com")
True
```

e-bol.net

ANHANG B

Mathematische Formeln

Im Buch habe ich eine Reihe von mathematischen Konzepten vorgestellt. Dieser Anhang behandelt ausgewählte Konzepte – mit Beschreibung, relevanten Formeln und entsprechendem Code zum Umsetzen.

Euklidischer Abstand

Der *euklidische Abstand* ermittelt die Entfernung zwischen zwei Punkten im multidimensionalen Raum. Dabei ist dies der Abstand, den Sie mit einem Lineal ermitteln würden. Werden die Punkte als (p_1, p_2, p_3, p_4, ...) und (q_1, q_2, q_3, q_4, ...) geschrieben, kann die Formel für den euklidischen Abstand wie in Gleichung B-1 ausgedrückt werden.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Gleichung B-1: Euklidischer Abstand

Eine einfache Implementierung dieser Formel sieht so aus:

```
def euclidean(p,q):
  sumSq=0.0

  # Quadrate der Differenzen addieren.
  for i in range(len(p)):
    sumSq+=(p[i]-q[i])**2

  # Wurzel ziehen.
  return (sumSq**0.5)
```

Der euklidische Abstand wird an vielen Stellen im Buch genutzt, um zu bestimmen, wie ähnlich sich zwei Elemente sind.

e-bol.net

Korrelationskoeffizient nach Pearson

Der *Pearson-Korrelationskoeffizient* ist ein Wert, der angibt, wie stark zwei Variablen korreliert sind. Er kann Werte zwischen 1 und –1 annehmen, wobei 1 dafür steht, dass die Variablen perfekt korreliert sind, 0 für keine Korrelation und –1 für eine perfekte inverse Korrelation.

Gleichung B-2 zeigt den Pearson-Korrelationskoeffizienten.

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{N}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{N}\right)}}$$

Gleichung B-2: Pearson-Korrelationskoeffizient

Dies kann durch folgenden Code implementiert werden:

```
def pearson(x,y):
  n=len(x)
  vals=range(n)

  # einfache Summen
  sumx=sum([float(x[i]) for i in vals])
  sumy=sum([float(y[i]) for i in vals])

  # Quadratsummen
  sumxSq=sum([x[i]**2.0 for i in vals])
  sumySq=sum([y[i]**2.0 for i in vals])

  # Summe der Produkte
  pSum=sum([x[i]*y[i] for i in vals])

  # Berechnung des Pearson-Korrelationskoeffizienten
  num=pSum-(sumx*sumy/n)
  den=((sumxSq-pow(sumx,2)/n)*(sumySq-pow(sumy,2)/n))**.5
  if den==0: return 0

  r=num/den

  return r
```

Wir haben den Pearson-Korrelationskoeffizienten in Kapitel 2 genutzt, um die Ähnlichkeit der Vorlieben zweier Personen zu bestimmen.

e-bol.net

Gewichtetes Mittel

Das *gewichtete Mittel* ist eine Form des Durchschnitts, die eine Gewichtung für jede Beobachtung hat, ie in das Mittel einfließt. Es wird in diesem Buch verwendet, um numerische Vorhersagen basierend auf Similarity Scores zu treffen. Das gewichtete Mittel hat die in Gleichung B-3 gezeigte Formel, wobei $x_1 \ldots x_n$ die Beobachtungen und $w_1 \ldots w_n$ die Gewichtungen sind.

$$\bar{x} = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}{w_1 + w_2 + \ldots + w_n}$$

Gleichung B-3: Gewichtetes Mittel

Eine einfache Implementierung dieser Formel erwartet eine Liste mit Werten und Gewichtungen:

```
def weightedmean(x,w):
  num=sum([x[i]*w[i] for i in range(len(w))])
  den=sum([w[i] for i in range(len(w))])

  return num/den
```

In Kapitel 2 wurde das gewichtete Mittel verwendet, um vorherzusagen, wie sehr jemand einen Film mögen wird. Das geschieht, indem man eine durchschnittliche Bewertung anderer Personen ermittelt, die danach gewichtet sind, wie ähnlich der jeweilige Geschmack dieser Person Ihrem Geschmack ist. In Kapitel 8 wurden gewichtete Mittel verwendet, um Preise vorherzusagen.

Tanimoto-Koeffizient

Der *Tanimoto-Koeffizient* ist eine Kennzahl, die die Ähnlichkeit zweier Sets beschreibt. Sie wird in diesem Buch genutzt, um zu berechnen, wie ähnlich sich zwei Elemente sind – basierend auf ihren Eigenschaftslisten. Nehmen Sie zwei Sets A und B, die so aussehen:

A = [Shirt, Schuhe, Hosen, Socken]
B = [Shirt, Rock, Schuhe]

Dann ist die Schnittmenge, die ich hier als C bezeichnen will, [Shirt, Schuhe]. Der Tanimoto-Koeffizient ist in Gleichung B-4 beschrieben, wobei N_a die Anzahl der Elemente in A und N_b die Anzahl der Elemente in B ist. N_c ist die Anzahl der Elemente in C, der Schnittmenge. In diesem Fall ist der Tanimoto-Koeffizient 2 / (4 + 3 – 2) = 2 / 5 = 0,4.

Hier eine einfache Funktion, die zwei Listen erwartet und den Tanimoto-Koeffizienten berechnet:

e-bol.net

$$T = \frac{N_c}{(N_a + N_b - N_c)}$$

Gleichung B-4: Tanimoto-Koeffizient

```
def tanimoto(a,b):
  c=[v for v in a if v in b]
  return float(len(c))/(len(a)+len(b)-len(c))
```

Der Tanimoto-Koeffizient wird in Kapitel 3 genutzt, um bei der Clusteranalyse Ähnlichkeiten zwischen Personen zu ermitteln.

Bedingte Wahrscheinlichkeit

Eine *Wahrscheinlichkeit* ist eine Möglichkeit zu ermitteln, wie wahrscheinlich ein Ereignis eintritt. Sie wird normalerweise als $Pr(A) = x$ geschrieben (oder $P(A) = x$), wobei A ein Ereignis ist. Wenn wir zum Beispiel sagen, dass heute eine 20 %-Chance für Regen besteht, könnten wir zum Beispiel schreiben: $Pr(Regen) = 0{,}2$.

Würden wir bemerken, dass es schon bewölkt ist, könnten wir folgern, dass es eine höhere Regenwahrscheinlichkeit gibt. Dies wird als *bedingte Wahrscheinlichkeit* bezeichnet, also die Chance für A, wenn wir wissen, das B eingetreten ist. Dies wird als $Pr(A \mid B)$ geschrieben, in diesem Fall also $Pr(Regen \mid Bewölkt)$.

Die bedingte Wahrscheinlichkeit wird ermittelt, indem man die Wahrscheinlichkeit für beide Ereignisse gemeinsam durch die Wahrscheinlichkeit der gegebenen Bedingung teilt, wie in Gleichung B-5 zu sehen.

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)}$$

Gleichung B-5: Bedingte Wahrscheinlichkeit

Wenn es in 10 % der Fälle morgens bewölkt ist und später regnet und es in 25 % der Fälle morgens bewölkt ist, ergibt sich $Pr(Regen \mid Bewölkt) = 0{,}1 / 0{,}25 = 0{,}4$.

Da dies nur eine einfache Division ist, wird keine Funktion dafür erstellt. Die bedingte Wahrscheinlichkeit wird in Kapitel 6 zum Filtern von Dokumenten genutzt.

Gini-Index

Der *Gini-Index* (*Gini Impurity*) ist ein Wert, der angibt, wie »verschmutzt« ein Set ist. Wenn Sie eine Reihe von Elementen haben, wie zum Beispiel [A, A, B, B, B, C], dann beschreibt der Gini-Index, wie wahrscheinlich es ist, dass Sie einen Fehlgriff tätigen,

e-bol.net

wenn Sie ein Element nehmen und zufällig seine »Beschreibung« erraten wollen. Besteht das Set nur aus As, würden Sie immer A raten und nie falsch liegen – das Set wäre also komplett »sauber«.

Gleichung B-6 zeigt die Formel für den Gini-Index.

$$I_G(i) = 1 - \sum_{j=1}^{m} f(i,j)^2 = \sum_{j \neq k} f(i,j)f(i,k)$$

Gleichung B-6: Gini-Index

Diese Funktion erwartet eine Liste mit Elementen und berechnet den Gini-Index:

```
def giniimpurity(l):
  total=len(l)
  counts={}
  for item in l:
    counts.setdefault(item,0)
    counts[item]+=1

  imp=0
  for j in l:
    f1=float(counts[j])/total
    for k in l:
      if j==k: continue
      f2=float(counts[k])/total
      imp+=f1*f2
  return imp
```

In Kapitel 7 wird der Gini-Index in Entscheidungsbäumen verwendet, um zu bestimmen, ob eine Aufteilung eines Sets für mehr »Reinheit« sorgt.

Entropie

Die *Entropie* ist ein weiterer Weg zu beschreiben, wie gemischt ein Set ist. Sie kommt aus der Informationstheorie und misst die Menge an Unordnung in einem Set. Grob gesagt, beschreibt die Entropie, wie überraschend ein zufällig gewähltes Element aus dem Set ist. Wenn das gesamte Set aus As bestünde, wären Sie nie überrascht, ein A zu erhalten, daher wäre die Entropie 0. Die Formel ist in Gleichung B-7 angegeben.

$$H(X) = \sum_{i=1}^{n} p(x_i) log_2\left(\frac{1}{p(x_i)}\right) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

Gleichung B-7: Entropie

e-bol.net

Diese Funktion erwartet eine Liste mit Elementen und berechnet die Entropie:

```
def entropy(l):
  from math import log
  log2=lambda x:log(x)/log(2)

  total=len(l)
  counts={}
  for item in l:
    counts.setdefault(item,0)
    counts[item]+=1

  ent=0
  for i in counts:
    p=float(counts[i])/total
    ent-=p*log2(p)
  return ent
```

In Kapitel 7 wird die Entropie in Entscheidungsbäumen verwendet, um zu bestimmen, ob ein Aufteilen eines Sets die Unordnung reduziert.

Varianz

Die *Varianz* bestimmt, wie sehr sich eine Menge von Zahlen vom Durchschnittswert unterscheidet. Sie wird in der Statistik häufig genutzt, um zu messen, wie groß die Unterschiede in einer Reihe von Zahlen sind. Die Varianz wird berechnet, indem man die quadrierten Abstände jeder Zahl vom Durchschnitt mittelt (siehe Gleichung B-8).

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Gleichung B-8: Varianz

Das lässt sich einfach implementieren:

```
def variance(vals):
  mean=float(sum(vals))/len(vals)
  s=sum([(v-mean)**2 for v in vals])
  return s/len(vals)
```

In Kapitel 7 wird die Varianz in Regressionsbäumen genutzt, um zu bestimmen, wie man ein Set am besten unterteilt, um die Untermengen enger verteilt zu bekommen.

Gauß-Funktion

Die *Gauß-Funktion* ist die Wahrscheinlichkeitsdichte-Funktion der Normalverteilung. Sie wird in diesem Buch als Gewichtungsfunktion für gewichtete k-nächste

e-bol.net

Nachbarn genutzt, da sie mit einem hohen Wert beginnt und schnell fällt, aber die 0 niemals erreicht.

Die Formel für eine Gauß-Verteilung mit einer Varianz σ finden Sie in Gleichung B-9.

$$\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Gleichung B-9: Gauß-Funktion

Sie lässt sich als Zweizeiler implementieren:

```
import math
def gaussian(dist,sigma=10.0):
  exp=math.e**(-dist**2/(2*sigma**2))
  return (1/(sigma*(2*math.pi)**.5))*exp
```

In Kapitel 8 wird die Gauß-Funktion als mögliche Gewichtungsfunktion zum Aufbau einer numerischen Vorhersage-Engine vorgeschlagen.

Skalarprodukt

Das *Skalarprodukt* (auch Punktprodukt, *Dot Product*) ist eine Methode, um zwei Vektoren miteinander zu multiplizieren. Wenn Sie zwei Vektoren $a = (a_1, a_2, a_3, \ldots)$ und $b = (b_1, b_2, b_3, \ldots)$ haben, ist das Skalarprodukt wie in Gleichung B-10 definiert.

$$\mathbf{a} \bullet \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

Gleichung B-10: Skalarprodukt mit Komponenten

Das Skalarprodukt lässt sich einfach mit dieser Funktion implementieren:

```
def dotproduct(a,b):
  return sum([a[i]*b[i] for i in range(len(a))])
```

Wenn θ der Winkel zwischen zwei Vektoren ist, kann das Skalarprodukt auch wie in Gleichung B-11 definiert werden.

$$\mathbf{a} \bullet \mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$

Gleichung B-11: Skalarprodukt mit dem Winkel

Das bedeutet, dass Sie das Skalarprodukt nutzen können, um den Winkel zwischen zwei Vektoren zu bestimmen:

e-bol.net

```
from math import acos

# Berechnen der Größe eines Vektors.
def veclength(a):
  return sum([a[i]**2 for i in range(len(a))])**.5

# Berechnen des Winkels zwischen zwei Vektoren.
def angle(a,b):
  dp=dotproduct(a,b)
  la=veclength(a)
  lb=veclength(b)
  costheta=dp/(la*lb)
  return acos(costheta)
```

Skalarprodukte werden in Kapitel 9 genutzt, um die Winkel zwischen zwei Vektoren zu bestimmen und damit Elemente zu klassifizieren.

e-bol.net

Index

A

Abfragen 71–72
  Query-Funktion 71
Abkühlung
  definiert 106
  simuliert 106–108
Abstandsmetriken 33
  Definition 53
advancedclassify.py
  dotproduct (Funktion) 226
  dpclassify (Funktion) 227
  getlocation (Funktion) 231
  getoffset (Funktion) 237
  lineartrain (Funktion) 224
  loadnumerical (Funktion) 232
  matchcount (Funktion) 229
  matchrow (Klasse)
    loadmatch (Funktion) 220
  milesdistance (Funktion) 230, 231
  nonlinearclassify (Funktion) 237
  rbf (Funktion) 236
  scaledata (Funktion) 233
  scaleinput (Funktion) 233
  yesno (Funktion) 228
agesonly.csv (Datei) 220
Ähnlichkeit kennzeichnen 32
Akismet XIII, 153
akismettest.py 154
Aktienmarkt-Analyse 7
Aktienmarkt-Daten 270–275
  Ergebnisse anzeigen 273
  Handelsvolumen 270
  Handelsvolumen von Google 275
  Matrix vorbereiten 272
  NMF ausführen 273
  Schlusspreise 270
  Yahoo! Finance 271
Algorithmen 4
  andere Anwendungen des Lernens 6
  CART (*siehe* CART)
  elementbasiertes kollaboratives Filtern,
    Empfehlungsalgorithmen 31
  genetisch (*siehe* genetische Algorithmen)
  hierarchische Clusteranalyse 41
  kollaboratives Filtern 10
  Mass-and-Spring 124
  Matrix-Mathematik 263
  Merkmalsextraktion 253
  PageRank (*siehe* PageRank-Algorithmus)
  Stemming 69
  Zusammenfassung 306–339
    Bayes-Klassifikator 307–310
Amazon 6, 58
  Empfehlungs-Engine 9
angenommene Wahrscheinlichkeiten
  variieren 156
Anmeldungen vorhersagen 158–160
articlewords (Dictionary) 257
Atom-Feeds
  Parsen 340
  Wörter zählen 35–37
Audioscrobbler 32
aus Klicks lernen (*siehe* neuronales Netz,
  künstlich)
Ausgleichsgerade 14

e-bol.net

B

Backpropagation 90–93, 317
Baum parsen 280
Baum, rekursiv erstellen 166–168
Bäume (*siehe* Entscheidungsbäume)
Bayes-Klassifikator 156, 256, 307–310
  Merkmalskombination 310
  naiver 308
  Stärken und Schwächen 309
  Support-Vektor-Maschinen (SVMs) 249
  Training 307
Bayes-Theorem 139
Beautiful Soup 51, 342
  Anwendungsbeispiel 342
  Crawler 64
  Installation 342
bedingte Wahrscheinlichkeit 135, 350
  Bayes-Theorem 139
benutzerbasierte Effizienz 32
benutzerbasiertes Filtern
  versus elementbasiertes Filtern 31
benutzerbasiertes kollaboratives Filtern 26
Bergsteigen 103–106
  Random-Restart 105
Bewerten
  Abfragen 62
  inhaltsbasiert (*siehe* inhaltsbasierte Bewertung)
Bewertungsmetriken 78–83
  einfaches Zählen 78
  PageRank-Algorithmus 79–82
  Verwendung des Link-Texts 82
Bibliotheken von dritter Seite 340–346
  Beautiful Soup 342
  matplotlib 344
    Anwendungsbeispiel 345
    Installation 345
  NumPy 343
    Anwendungsbeispiel 344
    Installation auf anderen Plattformen 344
    Installation auf Windows 344
  pydelicious 345
    Anwendungsbeispiel 346
    Installation 345
  pysqlite 342
    Anwendungsbeispiel 343
    Installation auf anderen Plattformen 343
    Installation auf Windows 343
  Python Imaging Library (PIL) 341
    Anwendungsbeispiel 341
    Installation auf anderen Plattformen 341
    Installation auf Windows 341
  Universal Feed Parser 340
Biotechnologie 6
Black-Box-Methode 319
Blogs
  Clusteranalyse basierend auf den Wortfrequenzen 35
  Feeds
    filtern 148–151
    Wörter zählen 35–37
    (*siehe auch* Atom-Feeds; RSS-Feeds)
Boolesche Operationen 95
Breeding 109, 278, 291

C

CART (Classification and Regression Trees) 162–163
Chi-Quadrat-Verteilung 144
Clusteranalyse 33, 251, 257
  hierarchische (*siehe* hierarchische Clusteranalyse)
  K-Means-Clusteranalyse (*siehe* K-Means-Clusteranalyse)
  Spalte 45–47
  übliche Verwendung 33
  Wortvektoren (*siehe* Wortvektoren)
clusters.py 43
  bicluster (Klasse) 40
  draw2d (Funktion) 58
  drawdendrogram (Funktion) 44
  drawnode (Funktion) 45
  getheight (Funktion) 44
  hcluster (Funktion) 41
  printclust (Funktion) 42
  readfile (Funktion) 39
  rotatematrix (Funktion) 47
  scaledown (Funktion) 56
Clusterzentren 330
Cocktailparty-Problem 251
Crawler 63–66
  Beautiful Soup (API) 64
  Code 64–66
  urllib2 64
Crawling 61
Crossover 109, 278, 291

e-bol.net

D

Daten, Anzeigen in zwei Dimensionen 55–58
Daten-Clusteranalyse (*siehe* Clusteranalyse)
Datenmatrix 264
Dating-Sites 6
del.icio.us XIII, 345
  Link-Empfehlungen erstellen 22–26
    Datenmenge erstellen 24
    del.icio.us-API 23
    Nachbarn und Links empfehlen 25
deliciousrec.py
  fillItems (Funktion) 24
  initializeUserDict (Funktion) 24
Dendrogramm 38
  zeichnen 43–45
    drawnode (Funktion) 45
Diversität 296
docclass.py
  classifier (Klasse) 133, 152
    catcount (Methode) 148
    categories (Methode) 148
    classify (Methode) 141
    fcount (Methode) 147
    fisherclassifier (Methode) 143
    fprob (Methode) 135
    incc (Methode) 147
    incf (Methode) 147
    setdb (Methode) 147
    totalcount (Methode) 148
    train (Methode) 134
    weightedprob (Methode) 136
  fisherclassifier (Klasse)
    classify (Methode) 146
    fisherprob (Methode) 144
    setminimum (Methode) 145
  getwords (Funktion) 131
  naivebayes (Klasse) 138
    prob (Methode) 139
  sampletrain (Funktion) 135
dorm.py 118
  dormZielfunktion 121
  printsolution (Funktion) 120
downloadzebodata.py 51, 52

E

eBay XIII
eBay-API 210–217, 218
  Details für ein Element erhalten 214
  Entwicklerschlüssel 210
  Erstellen einer Preisvorhersage 216
  Quick Start Guide 210
  Suchen 212
  Verbindung einrichten 211
ebaypredict.py
  doSearch (Funktion) 212
  getCategory (Funktion) 213
  getHeaders (Funktion) 211
  getItem (Funktion) 215
  getSingleValue (Funktion) 211
  makeLaptopDataset (Funktion) 216
  sendrequest (Funktion) 211, 212
eingehende Links 78–83
  einfaches Zählen 78
  PageRank-Algorithmus 79–82
  Verwenden des Link-Texts 82
elementbasierte kollaborative Empfehlungs-Filter-Algorithmen 31
elementbasierte Lesezeichen-Filterung 32
elementbasiertes Filtern 26–29
  Datenmenge zum Vergleichen von Elementen 27–28
  Empfehlungen erhalten 28–29
  versus benutzerbasiertes Filtern 31
Elite 294
Empfehlungen basierend auf der Kaufhistorie 6
Empfehlungs-Engines 9–32
  ähnliche Benutzer finden 12–18
    euklidischer Abstand 12–14
    Kritiker bewerten 17
    Pearson-Korrelationskoeffizient 14–16
    verwendete Metrik 16
  elementbasiertes Filtern 26–29
    Datenmenge zum Vergleichen von Elementen 27–28
    Empfehlungen erhalten 28–29
  elementbasiertes Filtern versus benutzerbasiertes Filtern 31
  kollaboratives Filtern 10

e-bol.net

Link-Empfehlungen für del.icio.us 22–26
Datenmenge aufbauen 24
del.icio.us-API 23
Nachbarn und Links empfehlen 25
Objekte empfehlen 18–20
gewichtete Werte 18
passende Produkte 20–22
Übungen 32
Verwendung der MovieLens-Datenmenge 29–31
Vorlieben sammeln 10–12
verschachteltes Dictionary 10
Endpreis 270
Entfernungen mit Yahoo! Maps bestimmen 229
Entropie 165, 351
Code 351
Entscheidungsbäume 158–185
Anmeldungen vorhersagen 158–160
Anwendung 183–184
Anzeigen 168–171
grafisch 169–171
beste Aufteilung 163–166
CART-Algorithmus 162–163
Einführung 161–162
Entropie 165
Ergebnis-Wahrscheinlichkeiten 184
fehlende Daten 175–176, 185
fehlende Datenbereiche 184
Gini-Index 164
Hotness modellieren 180–183
Immobilienpreise modellieren 177–180
Zillow-API 177–180
in der realen Welt 172
Klassifizieren neuer Beobachtungen 171–173
Mehrfachaufteilung 185
Nachteile 184
numerische Ergebnisse 176
Pruning 173–174
rekursive Bäume 166–168
Training 162–163
Übungen 184
vorzeitiger Abbruch 184
Entscheidungsbaum-Klassifikator 222, 310–315
Interaktionen von Variablen und 314
Stärken und Schwächen 314
Training 311
Entscheidungsbaum-Modellierung 352
Entscheidungsgrenze 223
Erkennung finanzieller Betrügereien 7
euklidischer Abstand 226, 347
Code 347
k-nächste Nachbarn (kNN) 324
Score 12–14

F

Facebook 122
andere Facebook-Vorhersagen 250
Entwicklerschlüssel 244
Freundesdaten herunterladen 246
Session erstellen 244
Vermitteln mit 243–249
Vermittlungs-Datenmenge erstellen 247
facebook.py
arefriends (Funktion) 248
createtoken (Funktion) 245
fbsession (Klasse) 245
getfriends (Funktion) 246
getinfo (Methode) 246
getlogin (Funktion) 246
getsession (Funktion) 246
makedataset (Funktion) 248
makehash (Funktion) 245
sendrequest (Methode) 245
factorize (Funktion) 264
feedfilter.py 149
entryfeatures (Methode) 152
Feedforward-Algorithmus 87–90
Feed-Parser 254
Filtern
Dokumente (*siehe* Filtern von Dokumenten)
regelbasiert 131
Spam
Schwellenwert 140
Tipps 140
Filtern von Dokumenten 130–157
Akismet 153
angenommene Wahrscheinlichkeiten verändern 156
Blog-Feeds 148–151
Dokumente klassifizieren 131–132
Klassifikatoren trainieren 133–135
Fisher-Methode 142–146
Elemente klassifizieren 145
versus naiver Bayes-Klassifikator 142
Wahrscheinlichkeiten kombinieren 144

e-bol.net

künstliche Phrasenlänge 156
Merkmalserkennung verbessern 151–153
naiver Bayes-Klassifikator 137–142
Kategorie wählen 140
versus Fisher-Methode 142
neuronales Netz als Klassifikator 156
Persistieren trainierter Klassifikatoren 146–148
SQLite 147–148
Pr(Dokument) 156
Spam 131
Übungen 156
virtuelle Merkmale 156
Wahrscheinlichkeiten berechnen 135–137
angenommene Wahrscheinlichkeit 136
bedingte Wahrscheinlichkeit 135
Finanzmärkte 2
Fisher-Methode 142–146
Elemente klassifizieren 145
versus naiver Bayes-Filter 142
Wahrscheinlichkeiten kombinieren 144
Fitness-Funktion 278
Flugdaten 129
Flugsuchen 112–118
Future-Märkte 2

G

Gauß-Funktion 194, 352
Code 353
Gauß-gewichtete Summe 209
genaue Übereinstimmung 95
generatefeedvector.py 35, 36
getwords (Funktion) 36
Generation 108
genetische Algorithmen 108–112, 338
Crossover oder Breeding 109
Generation 108
Mutation 108
Population 108
versus genetische Programmierung 280
genetische Optimierung
Abbruchkriterien 128
genetische Programmierung 111, 277–305
Breeding 278
Crossover 278
Datentypen 302
Dictionaries 303
Listen 303
Objekte 303
Strings 303
Diversität 296
einfache Spiele 296–301
Grid War 296
jeder gegen jeden 298
Spielen gegen reale Personen 300
Elite 294
Erfolg messen 287
Erfolge 279
Ersetzungs-Mutation 304
erste Population erstellen 285
Evolution abbrechen 304
Fitness-Funktion 278
Funktionstypen 304
gegen reale Personen spielen 300
jeder gegen jeden 299
Knoten mit Datentypen 305
Lösung testen 286
Mutation 278
numerische Funktionen 301
Parse-Tree 280
Programme als Bäume 280–284
Programme mutieren 288–291
Python und 280–284
RoboCup 279
Speicher 302
Tic-Tac-Toe-Simulator 305
Überblick 278
Übungen 304
Umgebung aufbauen 293–296
verborgene Funktionen 304
versus genetische Algorithmen 280
weitere Möglichkeiten 301–304
zufälliges Crossover 304
Geocoding 230
API 230
gewichtete Bewertungen 18
gewichtete Nachbarn 192–196
Gauß-Funktion 194
gewichtete kNN 195
Glockenkurve 194
inverse Funktion 192
Subtraktions-Funktion 193
gewichteter Durchschnitt 195, 324

e-bol.net

gewichtetes Mittel 349
Code 349
Gewichtungs-Matrix 260
Gini-Index 164, 350
Code 351
globales Minimum 105, 337
Glockenkurve 194
Goldberg, David 10
Google 1, 3, 5
PageRank-Algorithmus (*siehe* PageRank-Algorithmus)
Google Blog Search 149
gp.py 281–286
buildhiddenset (Funktion) 287
constnode (Klasse) 281, 282
crossover (Funktion) 292
evolve (Funktion) 293, 296
fwrapper (Klasse) 281, 282
getrankfunction (Funktion) 294
gridgame (Funktion) 297
hiddenfunction (Funktion) 287
humanplayer (Funktion) 300
mutate (Funktion) 290
node (Klasse) 281, 282
display (Methode) 284
exampletree (Funktion) 283
makerandomtree (Funktion) 285
paramnode (Klasse) 281, 282
rankfunction (Funktion)
Breedingrate 294
Mutationsrate 294
popsize 294
probexp 294
probnew 294
scorefunction (Funktion) 288
tournament (Funktion) 299
Grade Inflation 15
Grid War 296
Spieler 304
GroupLens 29
Web-Site 31
Gruppen finden 33–60
Blog-Clusteranalyse 59
Cluster mit Vorlieben (*siehe* Interessencluster)
Daten-Clusteranalyse (*siehe* Clusteranalyse)
hierarchische Clusteranalyse (*siehe* hierarchische Clusteranalyse)
K-Means-Clusteranalyse (*siehe* K-Means-Clusteranalyse)
Manhattan-Abstand 59
multidimensionales Skalieren (*siehe* multidimensionales Skalieren)
SpaltenClusteranalyse (*siehe* Spalten-Clusteranalyse)
überwachtes versus unüberwachtes Lernen 34
Übungen 59
Gruppenreisekosten (Funktion) 128
Gruppenreiseplanung 97–98
Abflugszeiten 100
Auto-Mietdauer 100
Preis 100
Wartezeit 100
Zeit 100
Zielfunktion (*siehe* Zielfunktion)

H

Handelsverhalten 6
Handelsvolumen 270
heterogene Variablen 198–202
Skalieren der Dimensionen 200
hierarchische Clusteranalyse 37–43, 328–329
Algorithmus 41
ausführen 42
Ausgabe 43
Dendrogramm 38
Dichtheit 39
einzelne Cluster 40
Pearson-Korrelationskoeffizient 40
Holland, John 111
Hollywood Stock Exchange 6
Hot or Not XIII, 180–183
hotornot.py
getpeopledata (Funktion) 181
getrandomratings (Funktion) 180
HTML-Dokumente, Parser 342
hyperbolischer Tangens (tanh) (Funktion) 87

I

Immobilienpreise modellieren 177–180
Zillow-API 177–180
Independent Component Analysis 7

e-bol.net

Indexieren 62
Hinzufügen zum Index 69
Index aufbauen 66–70
Schema einrichten 67
Tabellen 67
Wörter auf einer Seite finden 68
inhaltsbasierte Bewertung 72–77
Normalisierung 74
Position des Dokuments 73
Wortabstände 73, 77
Wortfrequenz 73, 75
Intelligenz, sich entwickelnde 277–305
Interessencluster 50–54
Abstandsmetrik definieren 53
Beautiful Soup 51
Daten laden und vorbereiten 50
Ergebnisse der Clusteranalyse 53
Zebo 50
Zebo-Ergebnisse extrahieren 51
Interessenliste 229
inverse Chi-Quadrat-Funktion 144
inverse Funktion 192
IP-Adressen 156

J

Ja/Nein-Fragen 228
Jaccard-Koeffizient 17
jeder gegen jeden 298

K

kategorische Merkmale
Entfernungen mit Yahoo! Maps bestimmen 229
Interessenlisten 229
Ja/Nein-Fragen 228
Kayak XIII, 129
API 112, 118
Daten 114
firstChild 114
getElementsByTagName 114
kayak.py 114
createschedule (Funktion) 117
flightsearch (Funktion) 115
flightsearchresults (Funktion) 115
getkayaksession( ) (Funktion) 115
Kernel
andere im LIBSVM 250
beste Kernel-Parameter 249
Kernel-Methoden 219–249
verstehen 234
Kernel-Trick 236–238, 320
Radial-Basisfunktion 236
Klassifikatoren
Bayes (*siehe* Bayes-Klassifikator)
einfache lineare 224–228
Entscheidungsbäume 222–224
Entscheidungsbäume (*siehe* Entscheidungsbaum-Klassifikator)
naiver Bayes- (*siehe* naiver Bayes-Klassifikator)
neuronales Netz 156
Training 133–135
Trainingsdaten abspeichern 146–148
SQLite 147–148
überwacht 251
Klassifizieren
Bayes-Klassifikator 308
Dokumente 131–132
Klassifikatoren trainieren 133–135
Klick-Training-Netzwerk 83
K-Means-Clusteranalyse 48–50, 276, 330–331
Funktion 49
k-nächste Nachbarn (kNN) 188–192, 324–328
Ähnlichkeit definieren 190
Anzahl der Nachbarn 189
euklidischer Abstand 324
gewichteter Durchschnitt 324
Kreuzvalidierung 326
Skalierung und überflüssige Variablen 325
Stärken und Schwächen 327
Verwendung 217
kollaboratives Filtern 10
Algorithmen 10
Begriff zuerst nutzen 10
kollektive Intelligenz
Definition 2
Einführung 1–8
Kreuzvalidierung 196–198, 326
Leave-one-out 217
Testsets 196
Trainingssets 196
Zahlen quadrieren 197
Kreuzvalidierungs-Funktion 243
kumulative Wahrscheinlichkeit 207

e-bol.net

Künstliche Intelligenz (KI) 4
künstliches neuronales Netz (*siehe* neuronales Netz, künstliches)

L

Längen und Breiten zweier Punkte in Entfernungen in Meilen umwandeln 231
Last.fm 6
LIBSVM
andere LIBSVM-Kernel 250
Anwendungen 240
Beispiel-Session 241
Vermittler-Datenmenge und 242
LIBSVM-Bibliothek 322
Lieferketten-Optimierung 7
lineare Klassifikation 224–228
Skalarprodukte 226
Vektoren 226
LinkedIn 122
lokale Minima 105, 337

M

Manhattan-Abstand 17, 59
Marketing 7
maschinelles Lernen 4
Grenzen 5
maschinelles Sehen 7
Mass-and-Spring-Algorithmus 124
matchmaker.csv (Datei) 220
mathematische Formeln 347–354
bedingte Wahrscheinlichkeit 350
Entropie 351
euklidischer Abstand 347
Gauß-Funktion 352
gewichtetes Mittel 349
Gini-Index 350
Pearson-Korrelationskoeffizient 348
Skalarprodukt 353
Tanimoto-Koeffizient 349
Varianz 352
matplotlib 206, 344
Anwendungsbeispiel 345
Installation 345
Matrix, umwandeln in 255
Matrix-Mathematik 259–270
Algorithmus 263
Daten-Matrix 264
Ergebnisse anzeigen 266, 274
factorize (Funktion) 264
Faktorisieren 260
Matrix vorbereiten 272
Multiplicative Update Rules 264
Multiplizieren 259
NumPy 262
Transponieren 260
maximal trennende Hyperebene 239
mehrlagiges Perzeptron (MLP) 83, 315
Merkmale 307
Merkmalsextraktion 251–275
Nachrichten 252–256
Merkmalsextraktions-Algorithmus 253
Merkmals-Matrix 260
minidom 113–114
minidom-API 178
MovieLens, Datenmenge nutzen 29–31
multidimensionales Skalieren 55–58, 60, 332–334
Code 333
Funktion 56
Pearson-Korrelation 55
Multiplicative Update Rules 264
Mutation 108, 278, 288–291

N

Nachrichtenbretter 130
Nachrichtenquellen 252–255
naiver Bayes-Klassifikator 137–142, 308
Kategorie wählen 140
Stärken und Schwächen 309
versus Fisher-Methode 142
nationale Sicherheit 7
Netflix 1, 6
Netzwerk-Visualisierung 122–128
kreuzende Linien zählen 124
Layout-Problem 122–124
Netzwerke zeichnen 126
neuronale Netze 62, 315–319
Backpropagation und 317
Blackbox-Methode 319
Code nutzen 318
Klassifikator 156
Kombinationen aus Wörtern 316
künstliche 83–94
Backpropagation 90–93
Datenbank einrichten 85–87

e-bol.net

Feed Forward 87–90
Klick-trainiertes Netz entwerfen 83
mit Such-Engine verbinden 93
Test-Training 93
mehrlagiges Perzeptron 315
Stärken und Schwächen 319
Synapsen und 315
Training 317
newsfeatures.py 253
getarticlewords (Funktion) 254
makematrix (Funktion) 255
separatewords (Funktion) 254
shape (Funktion) 263
showarticles (Funktion) 267, 268
showfeatures (Funktion) 266, 269
stripHTML (Funktion) 254
transpose (Funktion) 262
Nicht-negative Matrix-Faktorisierung (NMF) 259–266, 334–337
Code nutzen 336
Faktorisierung 34
Update Rules 335
Ziel 335
nn.py
searchnet (Klasse) 85
generatehiddennode (Funktion) 87
getstrength (Methode) 86
setstrength (Methode) 86
nnmf.py
difZielfunktion 263
Normalisierung 74
numerische Vorhersagen 186
numpredict.py
createcostfunction (Funktion) 202
createhiddendataset (Funktion) 203
crossvalidate (Funktion) 197, 202
cumulativegraph (Funktion) 207
distance (Funktion) 191
dividedata (Funktion) 196
euclidian (Funktion) 190
gaussian (Funktion) 194
getdistances (Funktion) 191
inverseweight (Funktion) 193
knnestimate (Funktion) 191
probabilitygraph (Funktion) 208
probguess (Funktion) 205, 207
rescale (Funktion) 201
subtractweight (Funktion) 193
testalgorithm (Funktion) 197
weightedknn (Funktion) 195
wineprice (Funktion) 187
wineset1 (Funktion) 187
wineset2 (Funktion) 199
NumPy 262, 343
Anwendungsbeispiel 344
Installation auf anderen Plattformen 344
Installation auf Windows 344
Verwendung 262

O

öffentliche Nachrichtenbretter 130
Online-Techniken 327
Open Web-APIs XIII
Optimierung 96–129, 202, 218, 337–339
Ausgangspunkte für Abkühlung 128
Bergsteigen 103–106
Flugsuchen (*siehe* Flugsuchen)
genetische Algorithmen 108–112
Crossover oder Breeding 109
Generation 108
Mutation 108
Population 108
genetische Optimierung, Abbruchkriterien 128
Gruppenreise, Zielfunktion 128
Gruppenreiseplanung 97–98
Abflugzeit 100
Auto-Mietdauer 100
Preis 100
Wartezeit 100
Zeit 100
Zielfunktion (*siehe* Zielfunktion)
Lösungen repräsentieren 98–99
Netzwerk-Visualisierung 122–128
kreuzende Linien zählen 124
Layout-Problem 122–124
Netze zeichnen 126
schlecht funktionierende Gebiete 111
simulierte Abkühlung 106–108
Strafe für kleine Winkel 129
Studenten zusammenbringen 129
Übungen 128
Vorlieben 118–122
Ausführen 121
Studentenwohnheim 118–121
Zielfunktion 121
Zielfunktion *siehe* Zielfunktion
Zufallssuche 102–103

e-bol.net

optimization.py 97, 203
annealingoptimize (Funktion) 107
geneticoptimize (Funktion) 109
elite 110
maxiter 111
mutprob 110
popsize 110
getminutes (Funktion) 98
hillclimb (Funktion) 104
printschedule (Funktion) 99
randomoptimize (Funktion) 102
schedulecost (Funktion) 101

P

PageRank-Algorithmus 5, 79–82
Pandora 6
Pearson-Korrelation
hierarchische Clusteranalyse 40
multidimensionales Skalieren 55
Pearson-Korrelationskoeffizient 14–16, 348
Code 348
Pilgrim, Mark 340
Polynom-Transformation 322
poplib 156
Population 108, 278, 338
Diversität und 285
Porter-Stemmer-Algorithmus 69
Position im Dokument 73, 75
inhaltsbasierte Bewertung
Pr(Dokument) 156
Preis für Hin- und Rückflug 129
Preismodelle 186–218
Anzahl der Nachbarn optimieren 217
Attribute suchen 218
Beispiel-Datenmenge erstellen 187–188
Elementtypen 218
k-nächste Nachbarn (kNN) 188
Laptop-Datenmenge 218
Leave-one-out-Kreuzvalidierung 217
ss zum Anzeigen der Wahrscheinlichkeit variieren 218
Übungen 217
Variablen eleminieren 217
Produkt-Marketing 7
Prognosemarkt 6
pydelicious 345
Anwendungsbeispiel 346
Installation 345
pysqlite 66, 342
Anwendungsbeispiel 343
Importieren 147
Installation auf anderen Plattformen 343
Installation auf Windows 343
Python
Tipps XI
Vorteile von X
Python Imaging Library (PIL) 43, 341
Anwendungsbeispiel 341
Installation auf anderen Plattformen 341
Installation auf Windows 341
Python, genetische Programmierung und 280–284
Anzeigen von Programmen 284
Aufbau und Evaluation von Bäumen 283
kompletten Baum durchlaufen 281
Repräsentation von Bäumen 281–283

Q

Query-Schicht 84

R

Radial-Basisfunktion 236
Random 102
Random-Restart-Bergsteigen 105
recommendations.py 10
calculateSimilarItems (Funktion) 27
getRecommendations (Funktion) 19
getRecommendedItems (Funktion) 28
loadMovieLens (Funktion) 30
sim_distance (Funktion) 13
sim_pearson (Funktion) 16
topMatches (Funktion) 17
transformPrefs (Funktion) 21
regelbasierte Filter 131
rekursiver Baumaufbau 166–168
RoboCup 279
RSS-Feeds
filtern 148–151
parsen 340
Wörter zählen 35–37
Rückgabe einer bewerteten Liste mit Dokumenten aus einer Abfrage 62

S

searchengine.py
addtoindex (Funktion) 69
crawler (Klasse) 62, 65, 67

e-bol.net

createindextables (Funktion) 67
distancescore (Funktion) 77
frequencyscore (Funktion) 75
getentryid (Funktion) 69
getmatchrows (Funktion) 71
gettextonly (Funktion) 68
import-Anweisungen 64
inboundlinkscore (Funktion) 78
isindexed (Funktion) 66, 70
linktextscore (Funktion) 82
neuronales Netz importieren 94
normalization (Funktion) 74
searcher (Klasse) 73
nnscore (Funktion) 94
query (Methode) 93
searchnet (Klasse)
backPropagate (Funktion) 91
trainquery (Methode) 92
updatedatabase (Methode) 92
separatewords (Funktion) 68
searchindex.db 67, 70
selbstorganisierende Karten 34
Sigmoid-Funktion 88
simulierte Abkühlung 106–108, 338
Skalarprodukt 226, 320, 353
Code 353
Skalieren
der Dimensionen 200
Optimierung 202–203
von Daten 233–234
Skalierung und überflüssige Variablen 325
socialnetwork.py 123
crosscount (Funktion) 125
drawnetwork (Funktion) 126
Spalten-Clusteranalyse 45–47
SpamBayes-Plugin 142
Spam-Filter 131
Methode 5
Schwellenwert 140
Tipps 140
Spidering (*siehe* Crawler)
SQLite 66
Schnittstelle zur eingebetteten Datenbank 342
Tabellen 67
trainierte Klassifikatoren persistieren 147–148
Stemming-Algorithmus 69
stochastische Optimierung 96
stockfeatures.txt (Datei) 274
stockvolume.py 272, 273, 274
factorize (Funktion) 273
Strafe für zu kleine Winkel 129
Studenten aufteilen 129
Studentenwohnheim, Vorlieben 118–121
Stützvektoren 239
Subtraktions-Funktion 193
Suchen über eingehende Links 95
Such-Engines
Abfragen (*siehe* Abfragen)
Bewertungsmetriken (*siehe* Bewertungsmetriken)
Boolesche Operationen 95
Crawler (*siehe* Crawler)
Dokumentensuche (lang/kurz) 95
genaue Treffer 95
Indexieren (*siehe* Indexieren)
inhaltsbasierte Bewertung (*siehe* inhaltsbasierte Bewertung)
Suche nach eingehenden Links 95
Überblick 61
Übungen 94
vertikal 112
Wortfrequenz
Verzerrung 95
Worttrennung 94
Support-Vektor-Maschinen (SVMs) 219–249, 319–323
andere LIBSVM-Kernel 250
Bayes-Klassifikator 249
Interessenhierarchie 250
Kernel-Trick 320
LIBSVM 322
Modelle erstellen 249
Polynom-Transformation 322
Skalarprodukte 320
Stärken und Schwächen 323
Trennlinie optimieren 249
Übungen 249
Synapsen 315

T

Tanimoto-Ähnlichkeit 32
Tanimoto-Koeffizient 53, 349
Code 349
Techniken zum überwachten Lernen 306–328

e-bol.net

Techniken zum unüberwachten Lernen 328–334
Test-Sets 196
Training
  Bayes-Klassifikator 307
  Entscheidungsbaum-Klassifikator 311
  neuronale Netze 317
  Sets 196
Transponieren 260
treepredict.py 160
  buildtree (Funktion) 167
  classify (Funktion) 171
  decisionnode (Klasse) 161
  divideset (Funktion) 162
  drawnode (Funktion) 170
  drawtree (Funktion) 170
  entropy (Funktion) 165
  mdclassify (Funktion) 175
  printtree (Funktion) 168
  prune (Funktion) 173
  split_function 163
  uniquecounts (Funktion) 164
  variance (Funktion) 176

U

überwachte Klassifikation 251
überwachte Lern-Methoden 34
Umwandeln von Längen und Breiten zweier Punkte in Entfernungen in Meilen 231
unabhängige Merkmale 251–276
  Abbruchkriterien 276
  alternative Anzeigemethoden 276
  K-Means-Clusteranalyse 276
  Nachrichtenquellen 276
  Optimieren der Faktorisierung 276
  Übungen 276
ungleiche Verteilungen 203–210
  Wahrscheinlichkeiten zeichnen 206
  Wahrscheinlichkeitsdichte schätzen 204
Universal Feed Parser 35, 148, 340
unüberwachte Techniken 251
unüberwachtes Lernen 34
Update Rules 335
urllib2 64, 113
Usenet 130

V

Varianz 352
  Code 352
Vektoren 226
Vektorwinkel berechnen 354
Vermittler-Datenmenge 220–243
Vermittlerdatenset
  Daten skalieren 233–234
  Entscheidungsbaum-Algorithmus 222–224
  kategorische Merkmale 228–232
  LIBSVM, Anwenden auf 242
  neu erstellen 232
  Schwierigkeiten mit Daten 221
verschachteltes Dictionary 10
vertikale Such-Engine 112
Verwendungsmuster für Wörter 252
virtuelle Merkmale 156
Volltext-Such-Engines (*siehe* Such-Engines)

W

Wahrscheinlichkeiten 350
  angenommene Wahrscheinlichkeit 136
  Bayes-Theorem 139
  bedingte Wahrscheinlichkeit 135
  darstellen 207
  des gesamten Dokuments bei gegebener Klassifikation 138
  kombinieren 144
  naiver Bayes-Klassifikator (*siehe* naiver Bayes-Klassifikator)
Wikipedia 3, 64
wordlocation (Tabelle) 71, 72
Wortabstand 73, 77
Wörter, die häufig zusammen genutzt werden 46
Wortfrequenz 73, 75
  Verzerrung 95
Worttrennung 94
Wortvektoren 34–37
  Blogs basierend auf den Wortfrequenzen clustern 35
  Wörter im Feed zählen 35–37

X

xml.dom 113
XML-Dokumente, Parser 342

e-bol.net

Y

Yahoo! Anwendungsschlüssel 230
Yahoo! Finance 58, 271
Yahoo! Groups 130
Yahoo! Maps 230

Z

Zahlen quadrieren 197
Zebo 50
  Ergebnisse extrahieren 51
  Web-Site 50
Zielfunktion 99–102, 121, 337
  globales Minimum 337
  lokale Minima 337
zillow.py
  getaddressdata (Funktion) 178
  getpricelist (Funktion) 179
Zillow-API 177–180
Zufallssuche 102–103

e-bol.net

e-bol.net

Über den Autor

Toby Segaran ist Direktor der Softwareentwicklung bei Genstruct, einer Bio-Informatik-Firma. Dort entwirft er Algorithmen und wendet Techniken zum Data-Mining an, um die Mechanismen von Drogen besser zu verstehen. Zudem arbeitet er mit anderen Firmen und Open Source-Projekten zusammen, um ihnen dabei zu helfen, ihre gesammelten Daten zu analysieren und auszuwerten. Außerdem hat er diverse freie Webanwendungen gebaut, auch die bekannten *tasktoy* und *Lazybase*. Er fährt gern Snowboard und mag es, Wein zu verkosten. Sein Blog findet sich unter *blog.kiwitobes.com*. Er lebt in San Francisco.

Über den Übersetzer

Thomas Demmig ist Diplom-Physiker. Er arbeitet bei der SAP AG als Entwickler im Bereich CRM und hat in den letzten sechs Jahren einige Bücher aus den Bereichen Datenbanken, Programmierung und Betriebssysteme übersetzt. Zudem entstammen zwei Bücher seiner eigenen Feder: *MySQL lernen* (Addison-Wesley) und *Jetzt lerne ich LaTeX* (Markt + Technik). Wenn er seine Freizeit nicht gerade mit seiner Frau Annika und seinen zwei Töchtern Friederike und Henriette verbringt, liest er oder versucht sich am Klavier.

Kolophon

Die Tiere auf dem Cover von *Kollektive Intelligenz analysieren, programmieren und nutzen* sind Königspinguine (*Aptenodytes patagonicus*). Auch wenn sie nach Patagonien benannt sind, brüten sie nicht mehr in Südamerika – die letzte Kolonie wurde von Siedlern im 19. Jahrhundert zerstört. Heutzutage findet man diese Pinguine auf subantarktischen Inseln wie Prince Edward, Crozet, Macquarie und den Falkland-Inseln. Sie leben an Stränden und auf flachen Eisflächen nahe dem Meer. Königspinguine sind extrem gesellige Vögel – sie brüten in Kolonien von bis zu 10.000 Tieren und ziehen ihre Küken in Krippen auf.

Mit seinen 75 cm Größe und einem Gewicht von bis zu 13,5 kg ist der Königspinguin einer der größten Pinguine – nach seinem nahem Verwandten, dem Kaiserpinguin. Abgesehen von seiner Größe sind die wichtigsten Erkennungszeichen des Königspinguins seine orangefarbenen Partien am Kopf, die bis zu seinen silberfarbenen Brustfedern reichen. Die Pinguine haben einen glatten Körper und können an Land laufen, anstatt wie die Kaiserpinguine hüpfen zu müssen. Sie sind gut an das Meer angepasst, essen Fisch und Tintenfische und können bis zu 250 m tief tauchen – weitaus tiefer als die meisten anderen Pinguine. Da sich Männchen und Weibchen in Größe und Aussehen gleichen, unterscheidet man sie aufgrund ihrer Verhaltensweisen, wie zum Beispiel bei Balzritualen.

e-bol.net

Königspinguine bauen keine Nester – stattdessen werden die Eier unter ihrem Körper aufbewahrt, wo sie auf den Füßen der Eltern liegen. Kein anderer Vogel hat einen so langen Brützyklus wie diese Pinguine, die in drei Jahren nur zweimal brüten und jeweils ein einzelnes Küken aufziehen. Die Küken sind rund, braun und so kuschelig, dass frühe Entdecker dachten, sie hätten es mit einer völlig anderen Pinguin-Spezies zu tun, die sie als »Wollpinguine« bezeichneten. Mit einer weltweiten Population von zwei Millionen Paaren sind Königspinguine keine bedrohte Spezies, und die World Conservation Union hat sie in die Kategorie »Nicht gefährdet« eingestuft.

Der Umschlagsentwurf dieses Buchs basiert auf dem Reihenlayout von Edie Freedman und stammt von Karen Montgomery, die hierfür einen Stich aus *Wood's Animate Creation* verwendet hat. Das Coverlayout der deutschen Ausgabe wurde von Michael Oreal mit InDesign CS3 unter Verwendung der Schriftart ITC Garamond von Adobe erstellt. Als Textschrift verwenden wir die Linotype Birka, die Überschriftenschrift ist die Adobe Myriad Condensed und die Nichtproportionalschrift für Codes ist LucasFont's TheSans Mono Condensed. Die in diesem Buch enthaltenen Abbildungen stammen von Robert Romano und Jessamyn Read und wurden mit Adobe Photoshop CS und Macromedia FreeHand MX erzeugt.