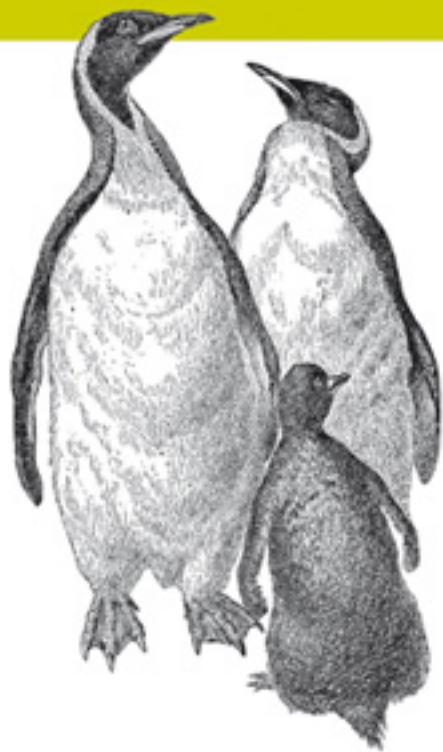


Lassen Sie Internet-Daten für sich arbeiten

**Deutsche
Ausgabe**

Kollektive Intelligenz

analysieren, programmieren & nutzen



O'REILLY®

Toby Segaran

Deutsche Übersetzung von Thomas Demmig

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

	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
	Merkmalerkennung verbessern	151
	Akismet	153
	Alternative Methoden	155
	Übungen	156

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

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

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.

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 traf 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,
```

```
'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}
```

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).

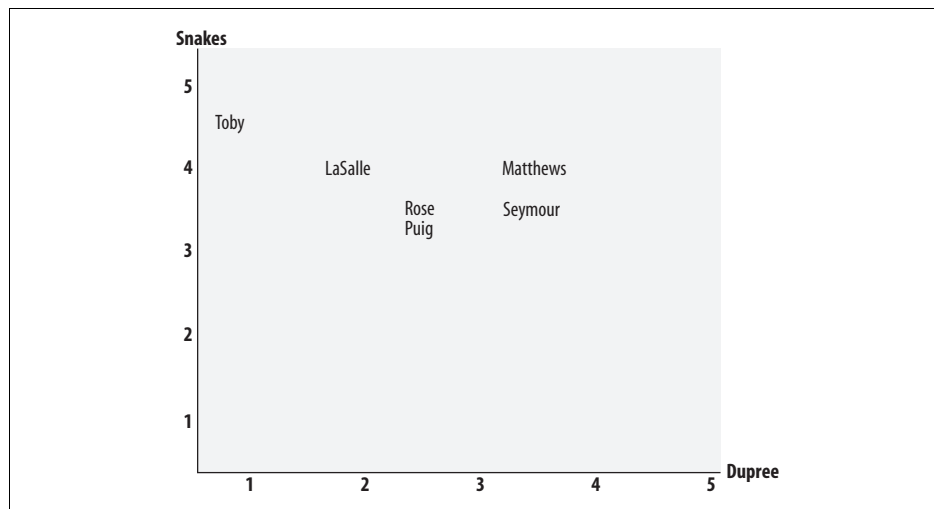


Abbildung 2-1: Personen im Preference Space

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.

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 abstands-basierten 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+sqrt(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
```

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.

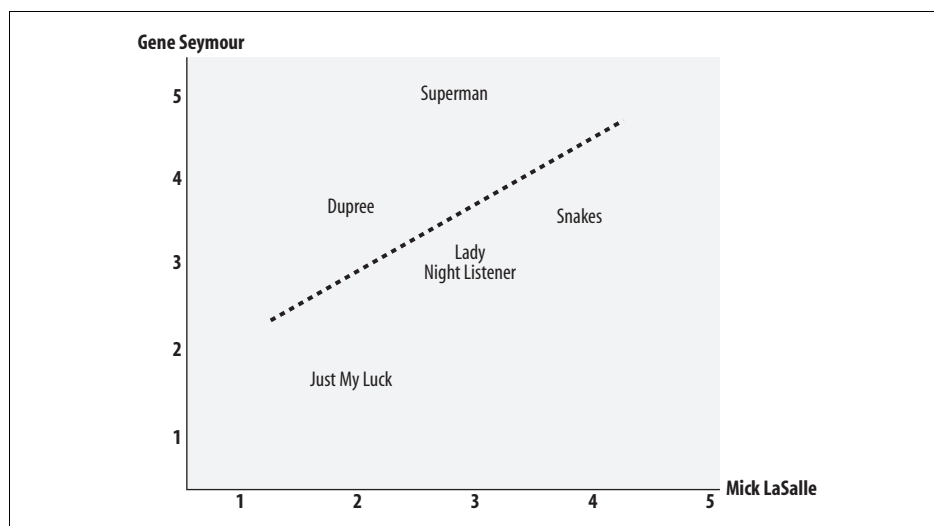


Abbildung 2-2: Der Vergleich zweier Filmkritiker mit einer Punktwolke

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-

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

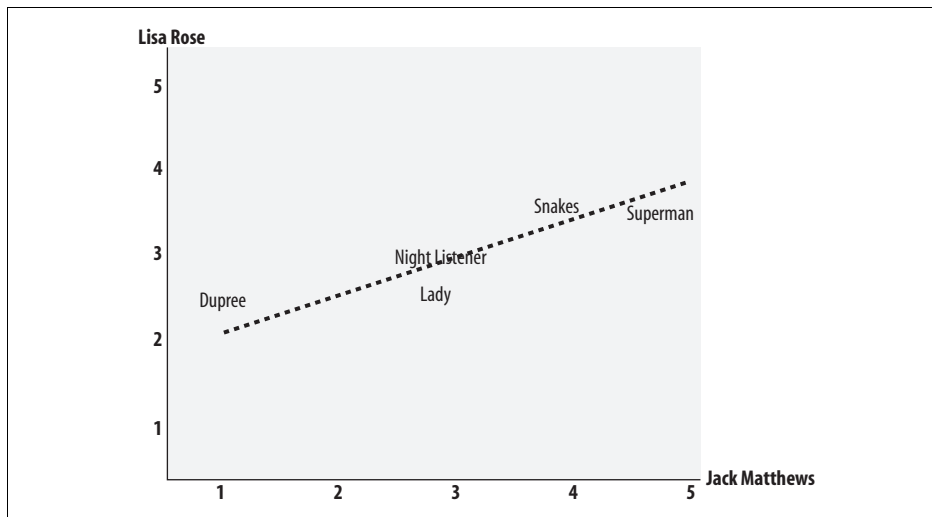


Abbildung 2-3: Zwei Kritiker mit einem hohen Korrelationskoeffizienten

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 Geschmäcker 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`:

```

# 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.

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')]
```

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.

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.

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.

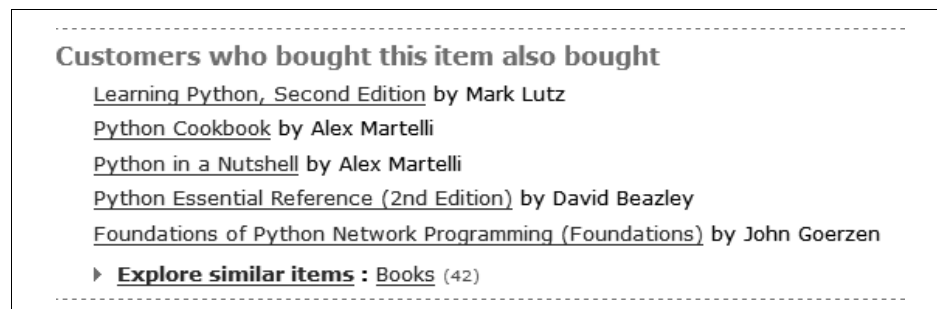


Abbildung 2-4: Amazon zeigt Produkte, die »Programming Python« ähneln

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,

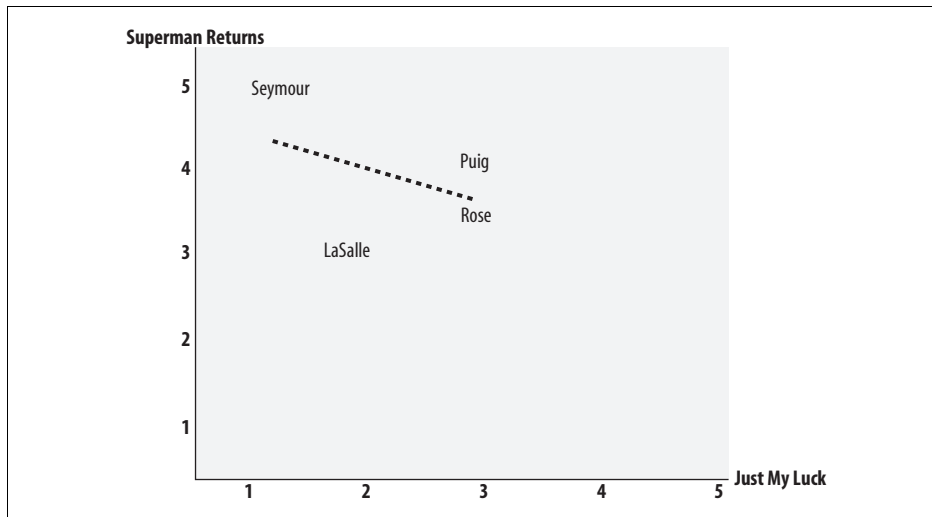


Abbildung 2-5: *Superman Returns* und *Just My Luck* haben eine negative Korrelation

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.



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'mmhale', '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.

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 für interessante 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
```

```
# 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'arturochoa'), (0.083, u'NickSmith'), (0.083,
u'MichaelDahl'), (0.050, u'zingoat')]
```

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:

```
>> 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.

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=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 Element-ähnlichkeiten halbwegs aktuell bleiben. Sie werden das häufiger tun müssen, solange die Anzahl der Benutzer noch klein und die Menge der Bewertungen gering

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={}
```

```

# 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.

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...]
```

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 erkaufte 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

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. *Audioscrobber*. Schauen Sie sich <http://www.audioscrobber.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.