Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Institut für Theoretische Informatik
Matthias Mitterreiter

Statistical Learning Theory Lab

Recommender System: k-Nearest-Neighbours

René Eduard Günter Gesele
Informatik
168039
02.10.1995
rene.gesele@uni-jena.de

Inhaltsverzeichnis

Implementierung eines Recommender Systems: k-Nearest-Neighbours

1 Recommender Systeme

Im Rahmen der Kundenzufriedenheit bieten große Internetanbieter heute alle Produktvorschläge an. Basierend auf Wissen über den Nutzer werden die bestmöglichen Produkte vorgeschlagen. Um diese Vorschläge machen zu können müssen zunächst die passenden Produkte gefunden werden. Dazu werden Recommender Systeme benötigt. Dabei kann man zwischen zwei Arten von Recommender System unterscheiden.

Beim “content filtering”- Ansatz werden über Nutzer und Produkt Profile erstellt, die diese gut charakterisieren. Dies kann zum Beispiel Alter, Gehalt und Geschlecht des Nutzers sein.

Der zweite Ansatz wird "collaborative filtering" genannt und verwendet als Wissen ausschließlich Nutzer-Produkt-Interaktionen. Diese sind meist in Form einer Bewertung gegeben, wobei auch schon der Kauf, das Ansehen oder Anhören des Produktes mitverwendet werden kann.

Im collaborative filtering gibt es hauptsächlich zwei Ansätze. Zum Einen gibt es latent factor models, die versuchen verdeckte Zusammenhänge zu schätzen. Diese Zusammenhänge (ca. 20 - 100) können zum Beispiel die Häufigkeit der Action, oder aber nicht interpretierbarer Natur sein.

Desweiteren gibt es nachbarschaftorientierte Algorithmen. Hat ein Nutzer verschiedene Produkte ähnlich bewertet wie eine Menge anderer Nutzer, dann werden Produktvorschläge aufgrund anderer gut bewerteter Produkte diese Menge von anderen Nutzern gemacht, die der Nutzer noch nicht bewertet hat. Werden ähnliche Nutzer gesucht wird es user-based genannt. Analog kann das auch für Produkte gemacht werden, item-based. Bei dem hier vorgestellte Algorithmus handelt es sich um einen user-based-Algorithmus, der die k-nearest-neighbours berechnet.

2 k-Nearest-Neighbours

Beim k-nearest-neighbours-Algorithmus werden die k ähnlichsten Objekte gesucht.
Dazu muss ein Distanz- oder Ähnlichkeitsmaß für die Objekte definiert werden.

Input: $O = \{c_0, \ldots, c_n\}$, $o \in O$, Anzahl k, Ähnlichkeitsmaß sim
Output: $c_{1\ldots k} \in$ nnc (Nearest-Neighbour-Candidates)

```
1.      nnc = {}
2.      for c in O:
3.              if |nnc| <=k:
4.                      nnc.add( (c , sim(o,c)) )
5.                      nnc.sort()
6.              else:
7.              if nnc.head[1] < sim(o,c)
8.                      nnc.head = (c, sim(o,c))
9.                      nnc.sort()
```

nnc ist eine sortierte Liste der k (falls schon k Objekte betrachtet wurden) besten Kandidaten, und deren Ähnlichkeit zu o, für die NN-Suche. Wird ein besserer Kandidat gefunden, ersetzt er head und nnc wird neu sortiert. Die Ähnlichkeit wird mit gespeichert. Nach dem alle c betrachtet wurden, enthält nnc die k ähnlichsten Objekte zu o.

3 Ähnlichkeitsmaße

Ähnlichkeits- oder Distanzmaße charakterisieren die Ähnlichkeit zweier Objekte, welche als Vektor gegeben sind. Während in vielen Anwendungen z.B. der euklidische Abstand Anwendung findet, wird bei Recommender Systemen häufig die Cosinus-Ähnlichkeit verwendet. Diese ist gegeben durch:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Die Ähnlichkeit wird durch den Winkel zwischen den beiden Vektoren A und B bestimmt. Sind A und B parallel, ist sim = 1; stehen A und B im rechten Winkel zueinander, ist sim = 0; Sind A und B antiparallel, ist sim = -1.

Die Cosinus-Ähnlichkeit ist unabhängig vom jeweiligen Mittelwert. Für Recommender Systeme heißt das, dass die Grundstimmung der Nutzer nicht mit aufgenommen wird. Falls zwei Nutzer unterschiedliche Zuordnungen für gleiche Ratings haben, wird dies nicht mit in der Ähnlichkeit berücksichtigt. Darum wird statt dem reinem Cosinus oft die Pearson-Ähnlichkeit verwendet:

$$\text{Sim}(u,v) = \text{Pearson}(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}}$$

Außerdem fließen nur die Produkte in die Gleichung mit ein, für die beide Nutzer eine Bewertung abgegeben haben. Von jedem Rating werden vor der Berechnung der Cosinus-Ähnlichkeit jeweils die Mittelwerte der zugehörigen Nutzer abgezogen.

Zuletzt braucht es eine Möglichkeit aufgrund der Ähnlichkeit und den k-nearest-neighbours einen Ratingwert für ein nicht bewertetes Item zu ermitteln. Dazu wird die Predictionfunktion verwendet, die wie folgt gegeben ist:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|}$$

Das so ermittelte Rating ist um den Mittelwert des Nutzers zentriert und es werden die k-nearest-neighbours verwendet die für das entsprechende Produkt eine Bewertung abgegeben haben.

4 Implementierung

Im folgenden werden die einzelnen Funktionen der Implementierung beschrieben.

4.1 Daten

Die Rating-Matrix ist in abstrakter Form gegeben. Spalte 0 bezeichnet die Zeile in der Rating Matrix, Spalte 1 die Spalte und Spalte 2 das zu der Nutzer-Produkt-Kombination gehörende Rating. Die Ratings nehmen ganzzahlige Werte von Null bis vier an.

4.2 Zusammenbau der dünnbesetzten Matrix

Der maximale Wert der ersten Spalte der Rating Matrix in abstrakter Form gibt die Anzahl der Zeilen und der maximale Wert der zweiten Spalte gibt die Anzahl der Spalten. Die gegebenen Ratings werden mit Eins addiert und übertragen, der Rest ist mit Nullen gefüllt.
Die Addition einer Eins wird durchgeführt, um die Nullen aus dem Rating von nicht bewerteten Einträgen zu unterscheiden.

4.3 desparse()

Da außer den Nutzer-Produkt-Interaktionen keine weitere Informationen gegeben sind, ist der Mittelwert der beste Schätzer.
Diese Funktion entfernt alle Zeilen und Spalten in denen ausschließlich Nullen stehen. Die Indizes der behaltenen Einträge werden in zwei Matrizen (eine für Zeilen, eine für Spalten) zurückgegeben. Zu Beginn des Algorithms enthalten diese Matrizen den Zeilen- bzw. Spaltenindex des jeweiligen Elements. Dann werden simultan die selben Zeilen und Spalten gelöscht wie in der dünnbesetzten Matrix. Dadurch bleiben die Indizes an den richtigen Stellen erhalten.

4.4 user_distance_matrix()

Da die Berechnung der Abstände bzw. Ähnlichkeit bei hochdimensionalen Daten relativ aufwendig ist, ist es zweckmäßig alle Abstände in einer Matrix zu speichern.

Diese Methode berechnet die obere Dreiecksmatrix der Abstände. Dabei entspricht die n-te Zeile und m-te Spalte dem Abstand zwischen den Nutzern n und m.
Um anschließend zeilenweise die Abstände eines Nutzers zu allen anderen auslesen zu können, wird die Matrix anschließend mit make_symmetric_matrix() symmetrisch gemacht.

4.5 matrix_of_knn()

Diese Funktion iteriert über alle Nutzer und berechnet mit knn() die nearest-neighbours. Dabei werden alle Nutzer ihrer Ähnlichkeit nach sortiert und mit ihrem zugehörigen Index in einem zweidimensionalen Array gespeichert. Der Index der ersten Achse entspricht dem Index der Nutzer in der dünnbesetzten Matrix und die Einträge sind die Indizes der Nachbarn.
Dieser Array funktioniert dann als Lookuptable um zu schauen, ob die jeweiligen Nachbarn das Produkt bewertet haben. Da der Array entlang der zweiten Achse sortiert ist, reicht es die top k Nachbarn mit Bewertungen für die Prediction zu verwenden.

4.6 prediction_matrix()

Diese Funktion iteriert über alle User und führt die Funktion predict_user_rating() für diesen aus.
predict_user_rating() iteriert über die verschiedenen Produkte und ermittelt die k-nearest-neighbours die das Produkt bewertet haben. Anschließend wird die oben unter Ähnlichkeitsmaße beschriebene Formel verwendet, um für jedes nicht bewertete Produkt eines Nutzers ein Rating zu berechnen. Zurückgegeben werden die einzelnen (vollständig bewerteten) Nutzer.
prediciton_matrix führt dann alle Nutzer wieder in einer Matrix zusammen.

4.7 rebuild_user_item_matrix(), fill_zeros_with_mean()

Mittels der Rückgabematrizen von desparse() kann mit rebuild_user_item_matrix() die ursprüngliche dünnbesetzte Matrix wieder hergestellt werden.
Anschließend können die wieder hinzugefügten Nullen mit fill_zeros_with_mean() durch den Mittelwert ersetzt werden.

4.8 get_predictions_for_test_data()

Diese Funktion füllt die dritte Spalte der Testdaten mit den Werten aus der Predictionmatrix. Hierbei wird noch von jedem Rating Eins subtrahiert, da diese ja zu Beginn beim Bau der Sparsematrix addiert wurde.

5 Ergebnisse

Die ersten Ergebnisse lagen unter dem Wert der Mittelwertschätzung (0.895). Dies hatte verschiedene Gründe.
Zunächst war der Zusammenbau nach dem desparse() fehlerhaft. Die Zuordnung der Prediction zu den Indizes konnte nicht korrekt vorgenommen werden, weil Zeilen und Spalten an falscher Stelle eingefügt wurden.
Der zweite Fehler war, dass die Pearsonähnlichkeit nicht korrekt implementiert war. Statt der Werte, für die beide Nutzer ein Rating haben, wurden alle Ratings verwendet.
Der dritte Fehler war, nicht die k-nearest-neighbours mit einem eingetragenen Rating für das jeweilige Produkt zu verwenden, sondern immer die gleichen k Nachbarn.

Da die ersten beiden Fehler unter Einschluss des dritten passiert sind (und dieser wahrscheinlich der schwerwiegendste ist), ist es schwierig eine Aussage über deren Wirkung zu treffen.

Der vierte Fehler war den Mittelwert der Nutzervektoren über alle Elemente zu berechnen. Dies hat bei den dünnbesetzten Vektoren, die hier vorkommen, den Effekt, dass der Mittelwert wesentlich tiefer ist, als das mittlere Rating.
Nach dem Fix der hier aufgeführten Werte wurde ein Fehler von 1.028 errechnet.

6 Diskussion

Die erreichten Ergebnisse könnten durch miteinbinden der Produktähnlichkeiten verbessert werden. Die Methodik, die hier durchgeführt wurde, kann mit der invertierten Sparsenmatrix durchgeführt werden, um durch diese die Ratings für Produktähnlichkeiten zu erhalten. Die erhaltenen Ergebnisse können dann gemittelt werden.
Ein zweiter Schritt wäre die Kombination mit latent factor models.

7 Quellen

Charu C. Aggarwal (2016). Recommender Systems. Heidelberg: Springer

Yehuda Koren, Robert Bell & Chris Volinsky (2009). Matrix Factorization Techniques for Recommender Systems. IEEE Computer Society