Praktikum Data Mining

Recommender Systems

Oliver Fesseler Maria Florusß Stefan Seibert

Daniel Grießhaber

5. Juni 2014

Durchführung 1: Fiktive Filmbewertung

Ähnlichkeitsbestimmung

Bedeutung des Übergabeparameters `normed` in der Funktion `sim_euclid`:

Wenn der Parameter `normed` auf `True` gesetzt ist, wird dafür gesorgt, dass die Ähnlichkeit normalisiert wird. Ohne die Normalisierung würden zwei Personen sich immer unähnlicher mit einer steigenden Anzahl an Filmen, die beide bewertet haben, da die Differenzen zwischen den Bewertungen einfach nur zusammengezählt werden. Durch die Normalisierung wird dies behoben.

Implementierung der Funktion `topMatches(prefs, person, similarity)`:

```
1  def topMatches(prefs, person, similarity):
2
3      sim = {}
4      for candidate in prefs:
5          if candidate == person:
6              continue
7          sim[candidate] = similarity(prefs, person, candidate)
8
9      return sorted(sim.iteritems(), key=operator.itemgetter(1), reverse=
       True)
```

`topMatches()` berechnet die Ähnlichkeit für jede in `prefs` aufgelistete Person die Ähnlichkeit zu `person` und schreibt diese in ein Dictionary (Person : Ähnlichkeit). Dieses wird in eine absteigend sortierte Liste von Tupeln (Person, Ähnlichkeit) geschrieben, welche zurück gegeben wird.

Bei der Implementierung von `topMatches()` fiel uns auf, dass man als Similarity-Funktion nur `sim_euclid` und `sim_pearson` angeben kann, jedoch so keine Unterscheidung zwischen der normalisierten und der unnormalisierten Version von `sim_euclid` gemacht werden kann. Deshalb entschieden wir uns eine weitere Similarity-Funktion zur Verfügung zu

stellen:

```
def sim_euclid_normed(prefs, person1, person2):
    return sim_euclid(prefs, person1, person2, normed=True)
```

`sim_euclid_normed` ruft einfach die ursprüngliche Version von `sim_euclid` mit `normed = True` auf.

Vergleich der Ähnlichkeitsmaße:

Die Pearson-Ähnlichkeit eignet sich für persönliche Wertungen besser, da Nutzer unterschiedliche Wertungsgewohnheiten haben. So bewerten einige Nutzer Filme allgemein schlechter, andere geben kontrastreichere Bewertungen für gute und schlechte Filme ab. Die Pearson-Ähnlichkeit gleicht diese persönlichen Präferenzen aus.

Berechnung von Empfehlungen mit Userbasiertem Collaborative Filtering (UCF)

Implementierung der Funktion `getRecommendations(prefs,person,similarity)`:

- Anmerkung: Die Erklärung der Funktion bezieht sich auf das Beispiel mit den Filmbewertungen.

Wie in `topMatches()` wird zunächst ein Dictionary angelegt, dass jeder Person außer der 'Zielperson' einen Ähnlichkeitswert zuordnet.

Nun wird für jede andere Person festgestellt, welche Filme diese bewertet hat, die die 'Zielperson' noch nicht bewertet hat. Die Bewertung jedes Films wird in ein Dictionary geschrieben, dass jedem Film, den die 'Zielperson' noch nicht gesehen hat, die Summe aller Bewertungen dieses Films zuordnet. Gleichzeitig wird für jeden Film die Korrelation der Person, die ihn bewertet hat in einem Dictionary `kSum` festgehalten. So ordnet `kSum` jedem Film die Summe aller relevanten Korrelationen zu. Zuletzt wird für jeden Film die Summe aller Bewertungen durch die Summe aller relevanten Korrelationen geteilt um den tatsächlichen Empfehlungswert zu erhalten. Zurückgegeben wird eine absteigend sortierte Liste von Tupeln(Empfehlungswert, Film).

Beim Testen unserer Funktionen `getRecommendations()` stellten wir fest, dass die Similarity-Funktion `sim_pearson` nicht funktioniert, wenn eine Person nur einen Film gesehen hat, da hier die Standartabweichung berechnet wird und dies für nur einen Wert immer 0 liefert. So werden alle Korrelationen mit 0 angegeben. Um dies zu umgehen bieten sich die Similarity-Funktionen `sim_euclid` oder `sim_euclid_normed` an.

Berechnung von Empfehlungen mit Itembasiertem Collaborative Filtering (ICF)

Transponieren der `prefs`-Matrix:

Für die Umsetzung des ICF musste die bisher verwendete Bewertungsmatrix in ihre transponierte Form überführt werden. Dazu wurde die Funktion `transposeMatrix(prefs)` implementiert. Diese iteriert zunächst über alle User und schreibt die bewerteten Filme in ein Dictionary. Danach sind alle Filme die ein User bewertet haben kann im Dictionary vorhanden. Nun werden alle Filmbewertungen aus der ursprünglichen Matrix extrahiert und in der neuen Matrix eingefügt. Die transponierte Matrix wird zurück gegeben.

Die Transformation ist notwendig, da die Funktionen aus den oberen Aufgaben unverändert wiederverwertet werden sollten.

Implementierung der Funktion `calculateSimilarItems()`:

Die Funktion stellt fest wie ähnlich sich die Filme untereinander sind und gibt für jeden Film ein Dictionary zurück in dem die Ähnlichkeit zu allen anderen Filmen festgehalten ist.

Implementierung der Funktion `getRecommendedItems(prefs, name, similar_items)`:

Diese Funktion soll durch ein ICF, einem User Filme die er noch nicht gesehen hat empfehlen. Diese Empfehlung basiert auf der Bewertung der bisher gesehenen Filme.

Dafür werden zunächst alle Filme, die ein User bewertet hat in eine Liste geschrieben. Danach werden alle Filme die der User nicht bewertet hat ebenfalls in eine Liste geschrieben. Nun wird die Ähnlichkeit zwischen allen Filmen mit der Funktion `calculateSimilarItems()` berechnet. Über die nicht bewerteten Filme wird nun iteriert und die Ähnlichkeit zu

jedem vom User bewerteten Film ausgelesen. Diese Ähnlichkeit wird nun mit der Bewertung des Users multipliziert um eine userspezifische Gewichtung der Ähnlichkeit zu garantieren. Nun wird für jeden Film die Summe über diese gewichtete Ähnlichkeiten durch die Summe der ungewichteten Ähnlichkeiten geteilt. Dieser Wert beschreibt wie empfehlenswert der Film für den aktuellen User ist. Die Funktion gibt eine Liste von Tupeln (Film, Empfehlungswert) zurück.

Durchführung 2: last.fm Musikempfehlungen

Durchführung

Um die gewünschte last.fm Musikempfehlung durchführen zu können wurde im File `pylastRecTest.py` der Zugriff auf die last.fm API implementiert. Hierfür stand bereits die Datei `pylast.py` zur Verfügung. Nach Aufruf des Networkobjekts wurden die 10 Topfans der Band 'Soulfly' abgerufen. Anschließend die zurückgelieferten Userobjekte auf deren Usernamen reduziert.

Die beiden Methoden `createLastfmUserDict(fanNames)` und die oben beschriebene `getRecommendations(pref, person, similarity)` wurden im File `recommendations.py` implementiert und lediglich nach einem entsprechenden Import ausgeführt.

`createLastfmUserDict(fanNames)` liefert dabei eine Matrix von Usernames und Bands, die von mindestens einem User gehört werden, zurück, wobei der Value = 1 ist, falls der einzelne User die Band hört, anderenfalls 0.

Problematiken:

- Bei der Zuweisung aller Bands zu einem bestimmten User muss darauf geachtet werden, dass das Dictionary aller Bands (lstinlineallBands) als deep-copy übergeben wird. Anderenfalls werden die Zuweisungspaare 'User - Band gehört' mit jedem User für alle User überschrieben, sodass die resultierte Matrix nur 1en enthält.
- die Bandnamen sollten nur aus unicode-Zeichen bestehen, da sonst einige Bands

doppelt aufgeführt werden. Allerdings kann es dadurch zu Darstellungsproblemen kommen.

Feststellungen:

- Nur weil ein User in den Top Fans einer Band ist, kann daraus kein Rückschluss auf die Platzierung der Band in seinen persönlichen Top Bands gezogen werden.