Assignment 7

CS 532: Introduction to Web Science

Spring 2017

Grant Atkins

Finished on April 6, 2017

1

Question

1. Find 3 users who are closest to you in terms of age, gender, and occupation. For each of those 3 users:

- what are their top 3 favorite films?
- bottom 3 least favorite films?

Based on the movie values in those 6 tables (3 users X (favorite + least)), choose a user that you feel is most like you. Feel free to note any outliers (e.g., "I mostly identify with user 123, except I did not like ‘‘Ghost’’ at all").

This user is the "substitute you".

Answer

First approaching this problem I decided to use python 3.6 because the code written in the Programming Collective Intelligence book, by Toby Segaran, was also python [2]. Instead of manually picking out users from the data file provided, I wrote a script called **substituteMe.py** , shown in Listing 1, which filters all users by the gender Male "M", the occupation of "programmer" and the age range of greater than 20 and less than 23. This script found multiple users ages 21 and 23, but I decided to go with users with age 21 since I was just recently 21. This left me with 3 users with the ids: 603, 671 and 868.

To find their favorite and least favorite movies I added a function called *findMoviesMerge*, which matched each user's review to their movie names and return the bottom and top movies sorted by their ratings, there were of course other movies with rating 5 but I simply took the top and bottom 3 provided. Their tables for top 3 films and bottom 3 films are shown in Table 1, Table 2 and Table 3 respectively.

I identified that user 868 was the "substitute me". This user had excellent movie choice with me liking all of his top favorite movies as well as me also disliking the same movies. Who would make a live action version of Super Mario Bros?

Rank	**Top Favorite Movies**	**Rating**	**Least Favorite Movie**	**Rating**
1	Star Wars (1977)	5.0	Platoon (1986)	1.0
2	Blade Runner (1982)	5.0	Heat (1995)	1.0
3	Twelve Monkeys (1995)	5.0	Platoon (1986)	2.0

Table 1: User 603's favorite and least favorite movies

Rank	**Top Favorite Movies**	**Rating**	**Least Favorite Movie**	**Rating**
1	My Best Friend's Wedding (1997)	5.0	Cop Land (1997)	1.0
2	Walk in the Clouds, A (1995)	5.0	Long Kiss Goodnight, The (1996)	1.0
3	Terminator, The (1984)	5.0	Star Trek: First Contact (1996)	1.0

Table 2: User 671's favorite and least favorite movies

Rank	Top Favorite Movies	Rating	Least Favorite Movie	Rating
1	2001: A Space Odyssey (1968)	5.0	Lassie (1994)	1.0
2	Raiders of the Lost Ark (1981)	5.0	Super Mario Bros. (1993)	1.0
3	Empire Strikes Back, The (1980)	5.0	Herbie Rides Again (1974)	1.0

Table 3: User 868's favorite and least favorite movies

```
import csv
from pprint import pprint as pp


def chooseUsers():
    usersChosen = []
    with open("data/u.user") as f:
        reader = csv.reader(f, delimiter='|')

        for i in reader:
            age = int(i[1])
            # filter parameters
            if(i[2] == 'M' and i[3] == 'programmer' and
                    (age > 20 and age < 23)):
                usersChosen.append(i)

    return usersChosen


def findReviews(userIds):
    # pairs are user id -> array of reviews
    reviewDict = {}
    for i in userIds:
        reviewDict[i] = []
    with open("data/u.data", 'r') as f:

        for line in f:
            spl = line.split()
            for i in userIds:
                if(spl[0] == i):
                    reviewDict[i].append(spl)

    return reviewDict


def findMovie(movieId):
    with open("data/u.item", 'r') as f:
        reader = csv.reader(f, delimiter='|')
        for i in reader:
            itemId = i[0]
            if movieId == itemId:
```

```
42                  # id, name, URI
43                  return (i[0], i[1], i[4])
44
45
46  def findMoviesMerge(reviewDict):
47      userMovieDict = {}
48      for userId, reviews in reviewDict.items():
49
50          userMovieDict[userId] = {}
51          moviesReviewed = []
52          botMovies = []
53          topMovies = []
54          for r in reviews:
55              movieId = r[1]
56              rating = r[2]
57
58              movie = findMovie(movieId)
59              movie = tuple(rating) + movie
60              moviesReviewed.append(movie)
61
62          # botMovies.sort(key=lambda tup: tup[0])
63          moviesReviewed.sort(key=lambda tup: tup[0])
64          botMovies = moviesReviewed[:3]
65          topMovies = moviesReviewed[-3:]
66          userMovieDict[userId]["bottomMovies"] = botMovies
67          userMovieDict[userId]["topMovies"] = topMovies
68
69      return userMovieDict
70
71
72  if __name__ == "__main__":
73      chosenUsers = chooseUsers()
74      userIds = []
75      for i in chosenUsers:
76          userIds.append(i[0])
77
78      reviewDict = findReviews(userIds)
79      with open("data/closestUsers.txt", 'w') as f:
80          pp(findMoviesMerge(reviewDict), stream=f)
```

Listing 1: Python script for determining closest 3 users

2

Question

`2. Which 5 users are most correlated to the substitute you? Which 5 users are least correlated (i.e., negative correlation)?`

Answer

This question's answer relied heavily off the source code provided by the Programming Collective Intelligence book [2]. I created a script called **correlateUsers.py**, shown in Listing 2, which also is used later in questions 3 and 4. This question used the *loadMovieLens*, which generated the preferences, and *findCorrelations* methods which finds the Sim Pearson correlation coefficient between substitute me, user 868, and every other user based on their preferences. The results of this are shown in Tables 4 and 5 and were saved to **correlatedUsers.txt** in my Github repository [1].

User ID	Correlation
853	+1.0
857	+1.0
898	+1.0
625	+1.0
724	+1.0

Table 4: Most correlated users

User ID	Correlation
36	-1.0
404	-1.0
599	-1.0
628	-1.0
736	-0.9045340337332909

Table 5: Least correlated users

```
1
2  import csv
3  from math import sqrt
4  from pprint import pprint as pp
5
6
7  def sim_pearson(prefs, p1, p2):
8      '''
9      Returns the Pearson correlation coefficient for p1 and p2.
10     '''
11
12     # Get the list of mutually rated items
13     si = {}
14     for item in prefs[p1]:
15         if item in prefs[p2]:
16             si[item] = 1
17     # If they are no ratings in common, return 0
18     if len(si) == 0:
19         return 0
20     # Sum calculations
21     n = len(si)
22     # Sums of all the preferences
23     sum1 = sum([prefs[p1][it] for it in si])
24     sum2 = sum([prefs[p2][it] for it in si])
25     # Sums of the squares
26     sum1Sq = sum([pow(prefs[p1][it], 2) for it in si])
27     sum2Sq = sum([pow(prefs[p2][it], 2) for it in si])
28     # Sum of the products
29     pSum = sum([prefs[p1][it] * prefs[p2][it] for it in si])
30     # Calculate r (Pearson score)
31     num = pSum - sum1 * sum2 / n
32     den = sqrt((sum1Sq - pow(sum1, 2) / n) * (sum2Sq - pow(sum2,
           2) / n))
33     if den == 0:
34         return 0
35     r = num / den
36     return r
37
38
39 def findCorrelations(prefs):
40     most_correlated = []
41     least_correlated = []
42     correlations = {}
43     substituteMe = str(868)
44
45     users = {}
46     for line in open('data/u.user'):
47         (user, age, gender, job, zipcode) = line.split('|')
```

```
48          users.setdefault(user, {})
49          users[user] = {'age': age, 'gender': gender,
50                         'job': job, 'zipcode': zipcode}
51
52      for user, rest in users.items():
53          if substituteMe == user:
54              pass
55          else:
56              r = sim_pearson(prefs, substituteMe, user)
57              correlations[int(user)] = r
58
59      correlations = sorted(correlations.items(), key=lambda x: x
            [1])
60      pp(correlations)
61      least_correlated = correlations[:5]
62      most_correlated = correlations[-5:]
63
64      with open("data/correlatedUsers.txt", 'w') as f:
65          print("Most Correlated:", file=f)
66          pp(most_correlated, stream=f)
67          print("Least Correlated:", file=f)
68          pp(least_correlated, stream=f)
69
70
71  def transformPrefs(prefs):
72      '''
73      Transform the recommendations into a mapping where persons
            are described
74      with interest scores for a given title e.g. {title: person}
            instead of
75      {person: title}.
76      '''
77
78      result = {}
79      for person in prefs:
80          for item in prefs[person]:
81              result.setdefault(item, {})
82              # Flip item and person
83              result[item][person] = prefs[person][item]
84      return result
85
86
87  def getRecommendations(prefs, person, similarity=sim_pearson):
88      '''
89      Gets recommendations for a person by using a weighted
            average
90      of every other user's rankings
91      '''
92
```

```
93      totals = {}
94      simSums = {}
95      for other in prefs:
96          # Don't compare me to myself
97          if other == person:
98              continue
99          sim = similarity(prefs, person, other)
100         # Ignore scores of zero or lower
101         if sim <= 0:
102             continue
103         for item in prefs[other]:
104             # Only score movies I haven't seen yet
105             if item not in prefs[person] or prefs[person][item]
                    == 0:
106                 # Similarity * Score
107                 totals.setdefault(item, 0)
108                 # The final score is calculated by multiplying
                        each item by the
109                 #   similarity and adding these products
                        together
110                 totals[item] += prefs[other][item] * sim
111                 # Sum of similarities
112                 simSums.setdefault(item, 0)
113                 simSums[item] += sim
114     # Create the normalized list
115     rankings = [(total / simSums[item], item) for (item, total)
            in
116                 totals.items()]
117     # Return the sorted list
118     rankings.sort()
119
120     lowestRankings = rankings[:5]
121     topRankings = rankings[-5:]
122
123     return (lowestRankings, topRankings)
124
125
126 def topMatches(
127     prefs,
128     person,
129     n=5,
130     similarity=sim_pearson,
131 ):
132     '''
133     Returns the best matches for person from the prefs
            dictionary.
134     Number of results and similarity function are optional
            params.
135     '''
```

```
136
137     scores = [(similarity(prefs, person, other), other) for
            other in prefs
138               if other != person]
139     scores.sort()
140     # scores.reverse()
141     lowestScores = scores[:n]
142     highestScores = scores[-n:]
143     return (lowestScores, highestScores)
144
145
146 def loadMovieLens(path='./'):
147     # Get movie titles
148     movies = {}
149     for line in open(path + 'data/u.item', encoding="ISO
            -8859-1"):
150         (id, title) = line.split('|')[0:2]
151         movies[id] = title
152
153     # Load data
154     prefs = {}
155     for line in open(path + 'data/u.data'):
156         (user, movieid, rating, ts) = line.split('\t')
157         prefs.setdefault(user, {})
158         prefs[user][movies[movieid]] = float(rating)
159     return prefs
160
161
162 def saveScores(filename, lowScores, highScores):
163     with open(filename, 'w') as f:
164         print("Lowest Scores:", file=f)
165         pp(lowScores, stream=f)
166         print("Highest Scores:", file=f)
167         pp(highScores, stream=f)
168
169
170 if __name__ == "__main__":
171     # q2
172     prefs = loadMovieLens()
173     findCorrelations(prefs)
174     # 868 is substiteMe
175     # q3
176     getRecommendations(prefs, '868')
177     # q4
178     prefs = transformPrefs(prefs)
179     (lowestScore, highestScore) = topMatches(prefs, 'Citizen
            Kane (1941)')
180     saveScores("data/favoriteFilmCorrelation.txt", lowestScore,
            highestScore)
```

```
181     (lowestScore, highestScore) = topMatches(prefs, 'Mars
            Attacks! (1996)')
182     saveScores("data/worstFilmCorrelation.txt", lowestScore,
            highestScore)
```

Listing 2: Python script utilizing Programming Collective Intelligence's code

3

Question

```
3.  Compute ratings for all the films that the substitute you
have not seen.  Provide a list of the top 5 recommendations for films
that the substitute you should see.  Provide a list of the bottom
5 recommendations (i.e., films the substitute you is almost certain
to hate).
```

Answer

The answer for this question again relied heavily upon Programming Collective Intelligence's code since it provided the main method to solve this problem, the *getRecommendations* method, and is shown in Listing 2. This again utilized the Sim Pearson correlation coefficient between users and found all the movies my substitute user, user 868, hasn't seen. After a final score was calculated these scores were normalized on a 1 to 5 scale and sorted in order. I took the lowest 5 movies and the top 5 movies, again with some being the same weight I simply took the top 5 it provided. These are shown in Table 6 and 7

I was taken aback when my second most recommended movie was "Santa with Muscles (1996)." I also apparently hate any kind of Amityville horror movie. I'll have to look into Saint of Fort Washington.

Rank	**Movie**	**Rating**
1	Saint of Fort Washington, The (1993)	5.0
2	Santa with Muscles (1996)	5.0
3	Someone Else's America (1995)	5.0
4	The Deadly Cure (1996)	5.0
5	They Made Me a Criminal (1939)	5.0

Table 6: Most recommended movies

Rank	Movie	Rating
1	3 Ninjas: High Noon At Mega Mountain (1998)	1.0
2	Amityville 1992: It's About Time (1992)	1.0
3	Amityville: A New Generation (1993)	1.0
4	Amityville: Dollhouse (1996)	1.0
5	Babyfever (1994)	1.0

Table 7: Least recommended movies

4

Question

4. Choose your (the real you, not the substitute you) favorite and least favorite film from the data. For each film, generate a list of the top 5 most correlated and bottom 5 least correlated films. Based on your knowledge of the resulting films, do you agree with the results? In other words, do you personally like / dislike the resulting films?

Answer

NOT ATTEMPTED

5

Question

Extra credit (3 points)
5. Rank the 1,682 movies according to the 1997/1998 MovieLense data. Now rank the same 1,682 movies according to todays (March 2016) IMDB data (break ties based on # of users, for example: 7.2 with 10,000 raters > 7.2 with 9,000 raters).

Draw a graph, where each dot is a film (i.e., 1,682 dots). The x-axis is the MovieLense ranking and the y-axis is today's IMDB ranking.

What is Pearon's r for the two lists (along w/ the p-value)? Assuming the two user bases are interchangable (which might not be a good assumption), what does this say about the attitudes about the films after nearly 20 years?

Answer

NOT ATTEMPTED

6

Question

Extra credit (3 points)
6. Repeat #6, but IMDB data from approximately July 31, 2005. What is the cumulative error (in days) from the desired target day of July 31, 2005? For example, if 1 memento is from July 1, 2005 and another memento is from July 31, 2006, then the cumulative error for the two mementos is 30 days + 365 days = 385 days.

Note: the URIs in the MovieLens data redirect, be sure to use the final values as URI-Rs for the archives.

Answer

References

[1] Atkins, Grant. "CS532 Assignment 7 Repository" Github. N.p., 23 March 2017. Web. 23 March 2017.https://github.com/grantat/cs532-s17/tree/master/assignments/A7.

[2] Segaran, Toby. "Programming Collective Intelligence". O' Reilly, 2007. Web. 6 April 2017. http://shop.oreilly.com/product/9780596529321.do.