Assignment 6

CS 532: Introduction to Web Science

Spring 2018

Hrishikesh Gadkari

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

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

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

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

```
        Attacks! (1996)')
181     saveScores("data/worstFilmCorrelation.txt", lowestScore,
            highestScore)
182 2018 Git
```

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 substitute me's second most recommended movie was "Santa with Muscles (1996)." He also apparently hates 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

When looked through the movie data I saw one of my favorite movies of all time, Citizen Kane (1941) because of the amazing cinematography for its time, I chose this film as my favorite. For me least favorite film I chose Mars Attacks! (1996). This question used the *getRecommendations* method like in question 3, but this time I had to transform the preferences into a dictionary of movie as key and users as values. The method to do this was *transformPrefs*, which was also provided by the Programming Collective Intelligence book [2]. I then used the *topMatches* method to get the bottom and top most correlated films. The results are shown in Tables 8, 9, 10 and 11 with the data saved to a files named **favoriteFilmCorrelation.txt** and **worstFilmCorrelation.txt** on my Github repository [1].

Its difficult to comment on the results of this problem as I have never seen any of the films in the tables below, aside from Free Willy, but not even that sequel number. I imagine that the films that least correlate with Citizen Kane also align with my dislikes after looking them up on the IMDB website. Aside from that, I really can't comment on the rest of the films.

Rank	Movie	Correlation
1	Newton Boys, The (1998)	1.0
2	Palmetto (1998)	1.0
3	Savage Nights (Nuits fauves, Les) (1992)	1.0
4	Wild America (1997)	1.0
5	Heavy (1995)	1.0

Table 8: Citizen Kane highest correlated movies

Rank	Movie	Correlation
1	Maya Lin: A Strong Clear Vision (1994)	-1.0
2	Free Willy 3: The Rescue (1997)	-1.0
3	Bad Girls (1994)	-1.0
4	Big Bully (1996)	-1.0
5	Colonel Chabert, Le (1994)	-1.0

Table 9: Citizen Kane least correlated movies

Rank	Movie	Correlation
1	Winter Guest, The (1997)	1.0
2	Wonderful, Horrible Life of Leni Riefenstahl, The (1993)	1.0
3	Wooden Man's Bride, The (Wu Kui) (1994)	1.0
4	Kaspar Hauser (1993)	1.0
5	Palmetto (1998)	1.0

Table 10: Mars Attacks! highest correlated movies

Rank	Movie	Correlation
1	Across the Sea of Time (1995)	-1.0
2	8 Heads in a Duffel Bag (1997)	-1.0
3	Swan Princess, The (1994)	-1.0
4	8 Seconds (1994)	-1.0
5	Aparajito (1956)	-1.0

Table 11: Mars Attacks! least correlated movies

References

[1] Hrishi, Gadkari. "CS532 Assignment 6 Repository" Github. N.p., 21 March 2018. Web. 21 March 2018.https://github.com/Hrishi29/anwala.github.io/tree/master/Assignments/A6.

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