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:
```

```
                # id, name, URI
                return (i[0], i[1], i[4])


def findMoviesMerge(reviewDict):
    userMovieDict = {}
    for userId, reviews in reviewDict.items():

        userMovieDict[userId] = {}
        moviesReviewed = []
        botMovies = []
        topMovies = []
        for r in reviews:
            movieId = r[1]
            rating = r[2]

            movie = findMovie(movieId)
            movie = tuple(rating) + movie
            moviesReviewed.append(movie)

        # botMovies.sort(key=lambda tup: tup[0])
        moviesReviewed.sort(key=lambda tup: tup[0])
        botMovies = moviesReviewed[:3]
        topMovies = moviesReviewed[-3:]
        userMovieDict[userId]["bottomMovies"] = botMovies
        userMovieDict[userId]["topMovies"] = topMovies

    return userMovieDict


if __name__ == "__main__":
    chosenUsers = chooseUsers()
    userIds = []
    for i in chosenUsers:
        userIds.append(i[0])

    reviewDict = findReviews(userIds)
    with open("data/closestUsers.txt", 'w') as f:
        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

```python

import csv
from math import sqrt
from pprint import pprint as pp


def sim_pearson(prefs, p1, p2):
    '''
    Returns the Pearson correlation coefficient for p1 and p2.
    '''

    # Get the list of mutually rated items
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    # If they are no ratings in common, return 0
    if len(si) == 0:
        return 0
    # Sum calculations
    n = len(si)
    # Sums of all the preferences
    sum1 = sum([prefs[p1][it] for it in si])
    sum2 = sum([prefs[p2][it] for it in si])
    # Sums of the squares
    sum1Sq = sum([pow(prefs[p1][it], 2) for it in si])
    sum2Sq = sum([pow(prefs[p2][it], 2) for it in si])
    # Sum of the products
    pSum = sum([prefs[p1][it] * prefs[p2][it] for it in si])
    # Calculate r (Pearson score)
    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


def findCorrelations(prefs):
    most_correlated = []
    least_correlated = []
    correlations = {}
    substituteMe = str(868)

    users = {}
    for line in open('data/u.user'):
        (user, age, gender, job, zipcode) = line.split('|')
```

```
            users.setdefault(user, {})
            users[user] = {'age': age, 'gender': gender,
                           'job': job, 'zipcode': zipcode}

    for user, rest in users.items():
        if substituteMe == user:
            pass
        else:
            r = sim_pearson(prefs, substituteMe, user)
            correlations[int(user)] = r

    correlations = sorted(correlations.items(), key=lambda x: x[1])
    pp(correlations)
    least_correlated = correlations[:5]
    most_correlated = correlations[-5:]

    with open("data/correlatedUsers.txt", 'w') as f:
        print("Most Correlated:", file=f)
        pp(most_correlated, stream=f)
        print("Least Correlated:", file=f)
        pp(least_correlated, stream=f)


def transformPrefs(prefs):
    '''
    Transform the recommendations into a mapping where persons are described
    with interest scores for a given title e.g. {title: person} instead of
    {person: title}.
    '''

    result = {}
    for person in prefs:
        for item in prefs[person]:
            result.setdefault(item, {})
            # Flip item and person
            result[item][person] = prefs[person][item]
    return result


def getRecommendations(prefs, person, similarity=sim_pearson):
    '''
    Gets recommendations for a person by using a weighted average
    of every other user's rankings
    '''

```

```
    totals = {}
    simSums = {}
    for other in prefs:
        # Don't compare me to myself
        if other == person:
            continue
        sim = similarity(prefs, person, other)
        # Ignore scores of zero or lower
        if sim <= 0:
            continue
        for item in prefs[other]:
            # Only score movies I haven't seen yet
            if item not in prefs[person] or prefs[person][item] == 0:
                # Similarity * Score
                totals.setdefault(item, 0)
                # The final score is calculated by multiplying each item by the
                #   similarity and adding these products together
                totals[item] += prefs[other][item] * sim
                # Sum of similarities
                simSums.setdefault(item, 0)
                simSums[item] += sim
    # Create the normalized list
    rankings = [(total / simSums[item], item) for (item, total) in
                totals.items()]
    # Return the sorted list
    rankings.sort()

    lowestRankings = rankings[:5]
    topRankings = rankings[-5:]

    return (lowestRankings, topRankings)


def topMatches(
    prefs,
    person,
    n=5,
    similarity=sim_pearson,
):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
```

```python

    scores = [(similarity(prefs, person, other), other) for other in prefs
              if other != person]
    scores.sort()
    # scores.reverse()
    lowestScores = scores[:n]
    highestScores = scores[-n:]
    return (lowestScores, highestScores)


def loadMovieLens(path='./'):
    # Get movie titles
    movies = {}
    for line in open(path + 'data/u.item', encoding="ISO-8859-1"):
        (id, title) = line.split('|')[0:2]
        movies[id] = title

    # Load data
    prefs = {}
    for line in open(path + 'data/u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
    return prefs


def saveScores(filename, lowScores, highScores):
    with open(filename, 'w') as f:
        print("Lowest Scores:", file=f)
        pp(lowScores, stream=f)
        print("Highest Scores:", file=f)
        pp(highScores, stream=f)


if __name__ == "__main__":
    # q2
    prefs = loadMovieLens()
    findCorrelations(prefs)
    # 868 is substiteMe
    # q3
    getRecommendations(prefs, '868')
    # q4
    prefs = transformPrefs(prefs)
    (lowestScore, highestScore) = topMatches(prefs, 'Citizen Kane (1941)')
    saveScores("data/favoriteFilmCorrelation.txt", lowestScore, highestScore)
```

```
    (lowestScore, highestScore) = topMatches(prefs, 'Mars
        Attacks! (1996)')
    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 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

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

NOT ATTEMPTED

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.