CS532 Web Science: Assignment 7

Finished on March 31, 2016

Dr. Michael L. Nelson

Naina Sai Tipparti
ntippart@cs.odu.edu

Contents

Listings

List of Tables

Problem 1

Question

The goal of this project is to use the basic recommendation principles we have learned for user-collected data. You will modify the code given to you which performs movie recommendations from the MovieLense data sets.

The MovieLense data sets were collected by the GroupLens Research Project at the University of Minnesota during the seven-month period from September 19th, 1997 through April 22nd, 1998. We are using the "100k dataset"; available for download from:
http://grouplens.org/datasets/movielens/100k/

The code for reading from the u.data and u.item files and creating recommendations is described in the book Programming Collective Intelligence. Feel free to modify the PCI code to answer the following questions.

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 ll").

This user is the "substitute you".

1.1 Answer

Each question was answered by using some combination of the existing functions from `recommendations.py` and some functions that were added. All of the tables provided were created using the `tabulate` function (with some minor edits), which is shown in Listing 1. All of the code used can be found in Appendix A, which contains Listing **??**.

Question 1 was solved using the code in Listing 2, which utilizes the added `flatten` and `get_top` functions, which are found in Listing 3 and 4.

The results for Question 1 are shown in Tables 1, 2, 3 and 4.

```python
def tabulate(tuples, caption, label, colnames, output):
    output.write('\\begin{table}[h!]\n')
    output.write('\\centering\n')
    opts = '| ' + ' | '.join(['l' for i in xrange(len(tuples[0]))]) + ' |'
    output.write('\\begin{{tabular}}{{{0}}}\n'.format(opts))
    output.write('\\hline\n')
    header = ' & '.join(['{}' for i in xrange(len(tuples[0]))]).format(*colnames)
    output.write(header + ' \\\\\n\\hline\n')
    for item in tuples:
        temp = ' & '.join(['{}' for i in xrange(len(item))])
        output.write(temp.format(*item) + ' \\\\\n')
    output.write('\\hline\n\\end{tabular}\n')
    output.write('\\caption{{{0}}}\n'.format(caption))
    output.write('\\label{{tab:{0}}}\n'.format(label))
    output.write('\\end{table}\n\n')
```

Listing 1: tabulate function

```python
        getuser = {}
        user_filter = lambda x: x['gender'] == 'M' and x['job'] == 'student' and int(x['age']) == 23
        ratings = []
        for mid, movie in movies.iteritems():
            for user, user_ratings in prefs.iteritems():
                if user_filter(users[user]) and user_ratings.has_key(movies[mid]):
                    getuser.setdefault(int(user),{})
                    getuser[int(user)][movies[mid]]=float(user_ratings[movies[mid]])
                    ratings.append(user_ratings[movies[mid]])
        sorted_getuser = {}
        movie_sort = {}
        for user, user_movie in getuser.items():
            user_movie_sort = sorted(user_movie.items(), key=itemgetter(1), reverse=False)
            for title, rating in user_movie_sort[:3]:
                movie_sort.setdefault(title,rating)
            # print movie_sort
            sorted_getuser.setdefault(user, user_movie_sort[:3])
        sorted_avg_all = sorted(sorted_getuser.items(), key=itemgetter(0), reverse=False)
        # print sorted_avg_all
        top_raters = get_top(sorted_avg_all, key=lambda x,i: x[i][1][0])
        top_raters = [flatten(rater) for rater in sorted_avg_all]
        tabulate(top_raters, 'Users', 'user', ('User', 'Rating', 'Movie'), outfile)
        print "done with 1"
```

Listing 2: Question 1 code

```python
def get_top(sorted_list, key=lambda x, i: x[i][1], n=5):
    top = key(sorted_list, 0)
    top_items = []
    i = 0
    while i < n or key(sorted_list, i) == top:
        top_items.append(sorted_list[i])
        if i < n and key(sorted_list, i) != top:
            top = key(sorted_list, i)
        i += 1
    return top_items
```

Listing 3: get_top functions

```python
def flatten(tup, f=lambda x: (x[0], x[1][0][1], x[1][0][0])):
    return f(tup)
```

Listing 4: flatten function

1.1.1 3-Users Closest to Me in Terms of Age, Gender, and Occupation

Id	Age	Gender	Occupation
33	23	M	student
37	23	M	student
66	23	M	student
135	23	M	student
391	23	M	student
408	23	M	student
706	23	M	student
838	23	M	student

Table 1: All-Users Closest Match

1.1.2 3-Users Closest to Me Top 3 Favorite Films

User	Rating	Movie
33	5.0	Titanic (1997)
	4.0	Game, The (1997)
	4.0	Air Force One (1997)
37	5.0	Pulp Fiction (1994)
	5.0	Raiders of the Lost Ark (1981)
	5.0	Terminator, The (1984)
66	5.0	Return of the Jedi (1983)
	5.0	Air Force One (1997)
	5.0	Ransom (1996)
135	5.0	Silence of the Lambs, The (1991)
	4.0	Rear Window (1954)
	4.0	Liar Liar (1997)
391	5.0	Rear Window (1954)
	5.0	Magnificent Seven, The (1954)
	5.0	Blues Brothers, The (1980)
408	5.0	Liar Liar (1997)
	5.0	Lost Highway (1997)
	5.0	Everyone Says I Love You (1996)
706	5.0	Phenomenon (1996)
	5.0	Edge, The (1997)
	5.0	Star Wars (1977)
838	5.0	Bringing Up Baby (1938)
	5.0	Toy Story (1995)
	5.0	City of Lost Children, The (1995)

Table 2: Top 3 Favorite Films

1.1.3 3-Users Closest to Me Least 3 Favorite Films

User	Rating	Movie
33	3.0	Liar Liar (1997)
	3.0	Devil's Advocate, The (1997)
	3.0	Soul Food (1997)
37	1.0	Jurassic Park (1993)
	2.0	Twister (1996)
	2.0	Arrival, The (1996)
66	1.0	English Patient, The (1996)
	1.0	Muppet Treasure Island (1996)
	1.0	Excess Baggage (1997)
135	1.0	Tales from the Hood (1995)
	2.0	Jaws 2 (1978)
	2.0	Star Trek III: The Search for Spock (1984)
391	1.0	Mimic (1997)
	2.0	Star Trek: The Wrath of Khan (1982)
	2.0	Courage Under Fire (1996)
408	1.0	Mouse Hunt (1997)
	2.0	U Turn (1997)
	2.0	Conspiracy Theory (1997)
706	1.0	Game, The (1997)
	1.0	Fargo (1996)
	1.0	Crash (1996)
838	2.0	Mars Attacks! (1996)
	2.0	Independence Day (ID4) (1996)
	2.0	Air Force One (1997)

Table 3: Least 3 Favorite Films

1.1.4 Substitute Me

Selecting substitute me was a difficult choice. I couldn't relate to user 33, 37, 66, 391, 706, 838, despite the fact that I would rank "Titanic (1997)" similarly with a 5.

I am greatly removed from user 33, 37, 66, 391, 706, 838 all the most maximum ranked movies by 33, 37, 66, 391, 706, 838 would be in my minimum ranked movies determination.

I am not that nearby either with user 135, 408, the most elevated ranked movies will be in my 4 rate selection. So, User 135 was selected as substitute of me.

User	Rating	Movie
135	5.0	Silence of the Lambs, The (1991)
	4.0	Rear Window (1954)
	4.0	Liar Liar (1997)
	1.0	Tales from the Hood (1995)
	2.0	Jaws 2 (1978)
	2.0	Star Trek III: The Search for Spock (1984)

Table 4: User 135 was selected as substitute me

Problem 2

Question

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

2.1 Answer

2.1.1 5-Users Most and Least Correlated to the Substitute Me

Question 2 was solved using the code shown in Listing 5, with the `sim_pearson` and the `get_top` functions from Listings 6 and 3, respectively. The `flatten` function from Listing 4 was used to flatted the oddly arranged tuple that was created from the `sim_pearson` function.

```
        most_correlated = {}
        least_correlated = {}
        for user, rest in users.iteritems():
            if int(user) == 135:
                for user1, rest in users.iteritems():
                    if user == user1:
                        pass
                    else:
                        r = sim_pearson(prefs, user, user1)
                        if r == 1.0:
                            most_correlated.setdefault(int(user1), r)
                        if r == -1.0:
                            least_correlated.setdefault(int(user1), r)
        sorted_most_correlated = sorted(most_correlated.items(), key=itemgetter(0), reverse=
            False)[len(most_correlated) - 6:-1]
        sorted_least_correlated = sorted(least_correlated.items(), key=itemgetter(0),
            reverse=False)[len(least_correlated) - 6:-1]
        tabulate(sorted_most_correlated, 'Most Correlated', 'most', ('User', 'Pearson\'s r')
            , outfile)
        tabulate(sorted_least_correlated, 'Least Correlated', 'least', ('User', 'Pearson\'s
            r'), outfile)
        print "done with 2"
```

Listing 5: Question 2 code

```
def sim_pearson(prefs, p1, 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
    filename = p1
    outfile = open(filename, 'a')
    outfile.write(str(r) + "\t\t" + str(user1))
    outfile.write("\n")
    outfile.close()
    return r
```

Listing 6: sim_pearson function

The results for Question 2 are shown in Table 5 and 6.

User	Pearson's r
20	1.0
74	1.0
161	1.0
219	1.0
284	1.0

Table 5: Most Correlated Substitute Me

User	Pearson's r
26	-1.0
39	-1.0
55	-1.0
241	-1.0
300	-1.0

Table 6: Least Correlated Substitute Me

Problem 3

Question

Compute ratings for all the films that the substitute you hasn't 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

Question 3 was solved using the code shown in Listing 7, with the `movies_not_rated` function from Listings 8, respectively. The outcome is put away in variable `movie_rated1` which will contain the ranking of movies not yet seen by `substitute me` sorted from slightest no doubt sought recommended preference to the probably recommended preference.

The results for Question 3 are shown in Table 7 and 8.

```
        movie_rated1 = {}
        for user, rest in users.iteritems():
            if int(user) == 135:
                movie_rated1 = movies_not_rated(prefs, movies, user, genre)
        sorted_most_recomm = sorted(movie_rated1.items(), key=itemgetter(1), reverse=False)[
            len(movie_rated1) - 6:-1]
        sorted_least_recomm = sorted(movie_rated1.items(), key=itemgetter(1), reverse=True)[
            len(movie_rated1) - 6:-1]
        tabulate(sorted_most_recomm, 'Top 5 unseen movies recommendations', 'recommtop', ('
            Title', 'Recomm Ranking'), outfile)
        tabulate(sorted_least_recomm, 'Least 5 unseen movies recommendations', 'recommleast'
            , ('Title', 'Recomm Ranking'), outfile)
        print "done with 3"
```

Listing 7: Question 3 code

```
def movies_not_rated(prefs, movies, userid, movie_type):
    userRatings = prefs[userid]
    count = 0
    movie_rated = {}
    for id, title in movies.items():
        temp = 0
        for item, rating  in userRatings.items():
            if title == item:
                temp = 1
                movie_rated.setdefault(title, rating)
        for name, genre in movie_type.items():
            if title == name:
                if temp == 0:
                    try:
                        if genre == 'comedy':
                            movie_rated[title] = '3.0'
                            temp = 0
                        elif genre == 'action':
                            movie_rated[title] = '3.0'
                            temp = 0
                        elif genre == 'crime':
                            movie_rated[title] = '3.0'
                            temp = 0
                        elif genre == 'drama':
                            movie_rated[title] = '4.0'
                            temp = 0
```

```
                        elif genre == 'thriller':
                            movie_rated[title] = '5.0'
                            temp = 0
                        elif genre == 'scifi':
                            movie_rated[title] = '4.0'
                            temp = 0
                        elif genre == 'war':
                            movie_rated[title] = '3.0'
                            temp = 0
                        elif genre == 'mystery':
                            movie_rated[title] = '3.0'
                            temp = 0
                        else:
                            movie_rated[title] = '1.0'
                            temp = 0
                    except:
                        pass
    return movie_rated
```

Listing 8: movies_not_rated function

3.1.1 Top 5 Recommendations of Unseen films for Substitute Me

Title	Ranking
Stranger in the House (1997)	5.0
Killer (Bulletproof Heart) (1994)	5.0
Gaslight (1944)	5.0
Office Killer (1997)	5.0
Unforgettable (1996)	5.0

Table 7: Top 5 unseen movies recommendations

3.1.2 Bottom 5 Recommendations of Unseen films for Substitute Me

Title	Ranking
Jaws 2 (1978)	2.0
Star Trek III: The Search for Spock (1984)	2.0
Highlander (1986)	2.0
Hard Target (1993)	2.0
Tales From the Crypt Presents: Demon Knight (1995)	2.0

Table 8: Least 5 unseen movies recommendations

Problem 4

Question

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

Question 4 was solved using the code shown in Listing 10,, with the `pearson` and the `get_avg` functions from Listings 11 and 9, respectively. The `get_avg` function uses the `mean` function from the numpy python library [1].

The results for Question 4 are shown in Table 9 and 10.

```
def get_avg(prefs, mid, user_filter=lambda x: True):
    ratings = []
    for user, user_ratings in prefs.iteritems():
        if user_filter(users[user]) and user_ratings.has_key(movies[mid]):
            ratings.append(user_ratings[movies[mid]])
    if not ratings:
        return 0.0
    return mean(ratings)
```

Listing 9: get_avg functions

```
        fav_cor = {}
        nfav_cor ={}
        avg1 = get_avg(prefs, mfavs)
        for id, title in movies.items():
            avg2 = get_avg(prefs, id)
            r = pearson(avg1, avg2, prefs, mfavs, title)
            if r == 1.0:
                fav_cor.setdefault(str(title), str(r))
            elif r == -1.0:
                nfav_cor.setdefault(str(title), str(r))
        sorted_fav_correlated = sorted(fav_cor.items(), key=itemgetter(0), reverse=False)[
            len(fav_cor) - 6:-1]
        sorted_nfav_correlated = sorted(nfav_cor.items(), key=itemgetter(0), reverse=False)[
            len(nfav_cor) - 6:-1]
        tabulate(sorted_fav_correlated, 'Most Favourite Movie \textcolor{blue}{Godfather,
            The (1972)} Correlated', 'fav', ('Title', 'Pearson\'s r'), outfile)
        tabulate(sorted_nfav_correlated, 'Least Favourite Movie \textcolor{blue}{Godfather,
            The (1972)} Correlated', 'nfav', ('Title', 'Pearson\'s r'), outfile)
        print "done with 4"
```

Listing 10: Question 4 code

```
def pearson(avg1,avg2,prefs, movie1, movie2, movies):
    r={}
    r=0
    top = 0
```

```
    rbtm = 0
    lbtm = 0

    for user, movie in prefs.items():
        userRatings = prefs[user]
        for item, rating in userRatings.items():
            if movie1 == movies[item]:
                for user1, move in prefs.items():
                    userRatings = prefs[user1]
                    for item1, rating1 in userRatings.items():
                        if movie2 == movies[item1]:
                            if user1 == user:
                                top += (rating-avg1)*(rating1-avg2)
                                lbtm +=  (rating-avg1)*(rating-avg1)
                                rbtm += (rating1-avg2)*(rating1-avg2)

    if lbtm == 0 or rbtm == 0:
        r = 5
        return r
    else:
        r = top/(math.sqrt(lbtm)* math.sqrt(rbtm))
    return r
```

Listing 11: pearson movie function

Title	Person's r
Gang Related (1997)	1.0
Love and Other Catastrophes (1996)	1.0
Rhyme & Reason (1997)	1.0
When We Were Kings (1996)	1.0
Dark City (1998)	1.0

Table 9: Most Favourite Movie Godfather, The (1972) Correlated

Title	Person's r
Paris, France (1993)	-1.0
Hugo Pool (1997)	-1.0
Naked in New York (1994)	-1.0
Turbo: A Power Rangers Movie (1997)	-1.0
Heavyweights (1994)	-1.0

Table 10: Least Favourite Movie Godfather, The (1972) Correlated

The recommendations results are not amazing to me. As expressed in `User 135 was selected as substitute me`4 arrangement user 135 is nearest to me of course. Be that as it may, the outcomes are close in wording how I feel about `substitute me` ranking. I hate all movies not recommended and the recommended ones I would rate them somewhere around 2 and 3, like the most astounding ranking for user 135.

Problem 5

Question

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

Problem 6

Question

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] Numpy Developers. Python numpy module. http://www.numpy.org/, 2016.