Introduction to Web Science: Assignment #8

Dr. Nelson

Alexander Nwala

Thursday, November 13, 2014

Contents

Problem 1

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. It is available for download from http://www.grouplens.org/node/73

There are three files which we will use:

1. u.data: 100,000 ratings by 943 users on 1,682 movies. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of

```
user id | item id | rating | timestamp
```

The time stamps are unix seconds since 1/1/1970 UTC.

Example:

```
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
298 474 4   884182806
115 265 2   881171488
```

2. u.item: Information about the 1,682 movies. This is a tab separated list of

```
movie id | movie title | release date | video release date | IMDb URL
| unknown | Action | Adventure | Animation |Children's | Comedy | Crime
| Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery
| Romance | Sci-Fi | Thriller | War | Western |
```

The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data data set.
Example:

```
161|Top Gun (1986)|01-Jan-1986||http://us.imdb.comMtitleexact?
Top%20Gun%20(1986)|0|1|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0|0|0
162|On Golden Pond (1981)|01-Jan-1981|
|http://us.imdb.com/M/title-exact?On%20Golden%20Pond%20(1981)
|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0
163|Return of the Pink Panther, The (1974)|01-Jan-1974|
|http://us.imdb.com/M/titleexact?Return%20of%20the%20Pink%20Panther,
%20The%20(1974)|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0
```

3. u.user: Demographic information about the users. This is a tab separated list of:

```
user id | age | gender | occupation | zip code
```

Problem 1 continued on next page. . .

The user ids are the ones used in the u.data data set.
Example:

```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
```

The code for reading from the u.data and u.item files and creating recommendations is described in the book Programming Collective Intelligence (check email for more details). You are to modify recommendations.py to answer the following questions. Each question your program answers correctly will award you 1 point.

1. What 5 movies have the highest average ratings? Show the movies and their ratings sorted by their average ratings.

2. What 5 movies received the most ratings? Show the movies and the number of ratings sorted by number of ratings.

3. What 5 movies were rated the highest on average by women? Show the movies and their ratings sorted by ratings.

4. What 5 movies were rated the highest on average by men? Show the movies and their ratings sorted by ratings.

5. What movie received ratings most like Top Gun? Which movie received ratings that were least like Top Gun (negative correlation)?

6. Which 5 raters rated the most films? Show the raters' IDs and the number of films each rated.

7. Which 5 raters most agreed with each other? Show the raters' IDs and Pearson's r, sorted by r.

8. Which 5 raters most disagreed with each other (negative correlation)? Show the raters' IDs and Pearson's r, sorted by r.

9. What movie was rated highest on average by men over 40? By men under 40?

10. What movie was rated highest on average by women over 40? By women under 40?

Your output should clearly indicate the answers from the question you answered. Provide any relevant discussion.

SOLUTIONS

The solutions for these problem are outlined by the following steps:

1. **Solution 1 - Aggregate movie and rater data:** This was achieved by reading u.item and u.data and bringing items with common IDs into the same structure in order to produce the following lists

```
movies: <movie id, (movie title, ratingsArray)>

aggregateMovieData: [ (movie id, movie title,
average movie rating, number of ratings) ]

aggregateUserData: [ (rater id, gender, age,
{'movie id': movie rating}) ]
```

as outlined in Listing 1. Thereafter, a call to to getHighestRatedMovies(aggregateMovies, 10) (Listing 2.) produced Table 1.

Listing 1: Aggregate Movie and Rater data

```
#aggregate data
#output:
    #movies: <movie id, (movie title, ratingsArray)>
    #aggregateMovieData: [ (movie id, movie title, average movie rating, number of
         ratings) ]
    #aggregateUserData: [ (user id, gender, age, {'movie id': movie rating}) ]
def aggregateMovieAndUserData(path='./data/movielens/'):
    try:

        movies = {}
        aggregateMovieData = []
        for line in open(path + 'u.item'):
            (id, title) = line.split('|')[0:2]
            movies[id] = (title, [], -1)

        users = {}
        #populate user and movie data
        for line in open(path + 'u.data'):
            (user, movieid, rating, ts) = line.split('\t')

            user = user.strip()
            movieid = movieid.strip()
            rating = rating.strip()
            ts = ts.strip()

            users.setdefault(user, {})
            #movie title: movies[movieid][0]
            users[user][movieid] = float(rating)#key is movie title

            #movies[movieid]: (title, [ArrayOfRatings])
            #movies[movieid][0]: title
            #movies[movieid][1]: array of ratings
            movies[movieid][1].append(float(rating))

        #process movie data
        for movieId, tupleData in movies.items():
            averageRating = sum(tupleData[1]) / float(len(tupleData[1]))

            movietuples = (movieId, tupleData[0], averageRating, len(tupleData[1])
                )
```

Problem 1 continued on next page. . .

```
                aggregateMovieData.append(movietuples)


            aggregateUserData = []
            for line in open(path + 'u.user'):
                (userId, age, gender, occupation, zipCode) = line.split('|')

                #userTuples: <userId, gender, age, {'movie title': movie rating}>
                userTuples = (userId, gender, age, users[userId])
                aggregateUserData.append(userTuples)

        except:
            exc_type, exc_obj, exc_tb = sys.exc_info()
            fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
            print(fname, exc_tb.tb_lineno, sys.exc_info() )
            return

        return aggregateMovieData, aggregateUserData, movies
```

Listing 2: Get Highest Rated Movies

```
#get highest rated movies
def getHighestRatedMovies(movies, count):
    if( count > 0 and count < len(movies) ):
        movies = sorted(movies, key=lambda tup: tup[2], reverse=True)
        i = 1
        for movie in movies:
            print movie

            if( i == count ):
                break
            i = i + 1
```

2. **Solution 2:** Based upon Listing 1 (Aggregated data), a call to getMostRatedMovies(aggregateMovies, 5) (Listing 3.), produced Table 2.

Listing 3: Most Rated Movies

```
#get most rated movies
def getMostRatedMovies(movies, count):
    if( count > 0 and count < len(movies) ):
        movies = sorted(movies, key=lambda tup: tup[3], reverse=True)
        i = 1
        for movie in movies:
            print movie

            if( i == count ):
                break
            i = i + 1
```

3. **Solution 3 - Aggregate male and female data:** Similar to aggregating movie data, getFemaleAndMaleData() (Listing 4.) produces 2 lists - 1 for male, the other for female raters. Thereafter, the following call getHighestRatedMoviesByMenOrWomen(aggregateUsersFemale, 5, movies) (Listing 4.) produced Table 3.

Problem 1 continued on next page...

Table 1: Arbitrary Subset of Highest Rated Movies

ITEM	MOVIE-ID	MOVIE-NAME	RATING
1	1653	Entertaining Angels: The Dorothy Day Story (1996)	5.0
2	1500	Santa with Muscles (1996)	5.0
3	814	Great Day in Harlem, A (1994)	5.0
4	1122	They Made Me a Criminal (1939)	5.0
5	1536	Aiqing wansui (1994)	5.0

Table 2: Most Rated Movies

ITEM	MOVIE-ID	MOVIE-NAME	RATINGS COUNT
1	50	Star Wars (1977)	583
2	258	Contact (1997)	509
3	100	Fargo (1996)	508
4	181	Return of the Jedi (1983)	507
5	294	Liar Liar (1997)	485

Table 3: Arbitrary Subset of Highest Rated Movies by Women

ITEM	MOVIE-NAME	RATING
1	Everest (1998)	5.0
2	Someone Else's America (1995)	5.0
3	Foreign Correspondent (1940)	5.0
4	Visitors, The (Visiteurs, Les) (1993)	5.0
5	Stripes (1981)	5.0

Table 4: Arbitrary Subset of Highest Rated Movies by Men

ITEM	MOVIE-NAME	RATING
1	Entertaining Angels: The Dorothy Day Story (1996)	5.0
2	Letter From Death Row, A (1998)	5.0
3	Hugo Pool (1997) 5.0	5.0
4	Santa with Muscles (1996)	5.0
5	Leading Man, The (1996)	5.0

Problem 1 continued on next page...

Listing 4: Aggregate Male and Female and get Highest Rated

```
#input:
    #aggregateUserData: [ (user id, gender, Age, {'movie id': movie rating}) ]
#output:
    #aggregateUsersFemale: [ (user id, F, Age, {'movie id': movie rating}) ]
    #aggregateUsersMale: [ (user id, M, Age, {'movie id': movie rating}) ]
def getFemaleAndMaleData(aggregateUsers):
    aggregateUsersFemale = []
    aggregateUsersMale = []
    if( len(aggregateUsers) > 0 ):

        for user in aggregateUsers:
            if( user[1] == 'F' ):
                aggregateUsersFemale.append(user)
            else:
                aggregateUsersMale.append(user)

#output:
    #aggregateUsersFemale: [ (user id, F, {'movie id': movie rating}) ]
def getHighestRatedMoviesByMenOrWomen(aggregateUsersFemaleOrMale, count, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and len(movies) > 0)
        :
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []

        #aggregate ratings per movie
        for user in aggregateUsersFemaleOrMale:
            #userId: user[0]
            #gender: user[1]
            #Age: user[2]
            #<id,rating>: user[2]
            #tupleOfMovieRatingDictionary[]
            for movieid, rating in user[3].items():


                if( movieid in tupleOfMovieRatingDictionary ):
                    tupleOfMovieRatingDictionary[movieid].append(rating)
                else:
                    tupleOfMovieRatingDictionary[movieid] = []
                    tupleOfMovieRatingDictionary[movieid].append(rating)


        #average ratings
        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)

        #sort
        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)
```

Problem 1 continued on next page...

```
        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[ movieData[0] ][0], movieData[1]

            if( i == count ):
                break
            i = i + 1
```

4. **Solution 4:** Similar to solution 3, a call getHighestRatedMoviesByMenOrWomen(aggregateUsersMale, 5, movies) (Listing 4.) produced Table 4.

5. **Solution 5:**

   (a) **Populate movie and user dictionary:** This was done through the MovieLens's loadMovieLens() method (Listing 5.) with a single modification - the movies dictionary was indexed with the movie id instead of the movie title due to duplicate title entries.

   (b) **Calculate one vs rest similarity:** This was also done through a slightly modified MovieLen's calculateSimilarItems() method by the following calls:

```
# call for most correlated
mostSimilarToTopGun = calculateSimilarItems
( prefs=prefs,
  simMeasure=sim_pearson,
  n=5,
  reverseSimilarityFlag=True,
  transformMatrixFlag=True
)

# call for least correlated
leastSimilarToTopGun = calculateSimilarItems
( prefs=prefs,
  simMeasure=sim_pearson,
  n=5,
  reverseSimilarityFlag=False,
  transformMatrixFlag=True
)
```

   The modification allowed the option of passing the similarity measure as a function argument, as well as sorting the list in ascending or descending order. Finally the resultant similarity dictionary was indexed with Top Gun's movie id (161) to produce Table 5 and 6.

Listing 5: Movies Most/Least Similar to Top Gun

```
#one vs. rest similarity for everyone
calculateSimilarItems(prefs, simMeasure, n=10, reverseSimilarityFlag=True,
    transformMatrixFlag=True):
    '''
    Create a dictionary of items showing which other items they are
```

Table 5: Arbitrary Subset of Movies most Correlated to Top Gun

ITEM	MOVIE-NAME	CORRELATION COEFFICIENT
1	King of the Hill (1993)	1.0
2	Shiloh (1997)	1.0
3	Bhaji on the Beach (1993)	1.0
4	Dangerous Ground (1997)	1.0
5	Hear My Song (1991)	1.0

Table 6: Arbitrary Subset of Movies least Correlated to Top Gun

ITEM	MOVIE-NAME	CORRELATION COEFFICIENT
1	Babysitter, The (1995)	-1.0
2	Telling Lies in America (1997)	-1.0
3	Switchback (1997)	-1.0
4	Carried Away (1996)	-1.0
5	Two Much (1996)	-1.0

```
    most similar to.
    '''
    result = {}
    # Invert the preference matrix to be item-centric

    if( transformMatrixFlag ):
        #with transform: movie top similarity
        itemPrefs = transformPrefs(prefs)
    else:
        #without transform: user top similarity
        itemPrefs = prefs

    for item in itemPrefs:
        scores = topMatches(itemPrefs, item, n=n, similarity=simMeasure,
            reverseSimilarityFlag=reverseSimilarityFlag)

        result[item] = scores
    return result
```

6. **Solution 6 - get highest movie rater:** This was achieved through a simple counting function (Listing 6.), which produced Table 7.

Table 7: Highest Movie Raters

ITEM	RATER-ID	No. RATINGS
1	405	737
2	655	685
3	13	636
4	450	540
5	276	518

Problem 1 continued on next page. . .

Listing 6: Highest Movie Raters

```
#get highest movie raters
#input:
    #aggregateUsers: [ (user id, gender, Age, {'movie_title': movie rating}) ]
def getHighestMovieRaters(aggregateUsers, count):

    if( len(aggregateUsers) > 0 and count > 0 ):
        raterIDRatedMoviesCount = []

        for rater in aggregateUsers:

            #raterTuple: rater id, number of movies rated
            raterTuple = (rater[0], len(rater[3]))
            raterIDRatedMoviesCount.append(raterTuple)

        #sort
        raterIDRatedMoviesCount = sorted(raterIDRatedMoviesCount, key=lambda tup:
            tup[1], reverse=True)

        i = 1
        for rater in raterIDRatedMoviesCount:
            print rater[0], rater[1]

            if( i == count ):
                break
            i = i + 1
```

7. **Solution 7:**

   (a) **Generate distance matrix:** This was achieved through a call to calculateSimilarItems() by passing a Euclidean distance (sim_distance) metric argument (Listing 5.):

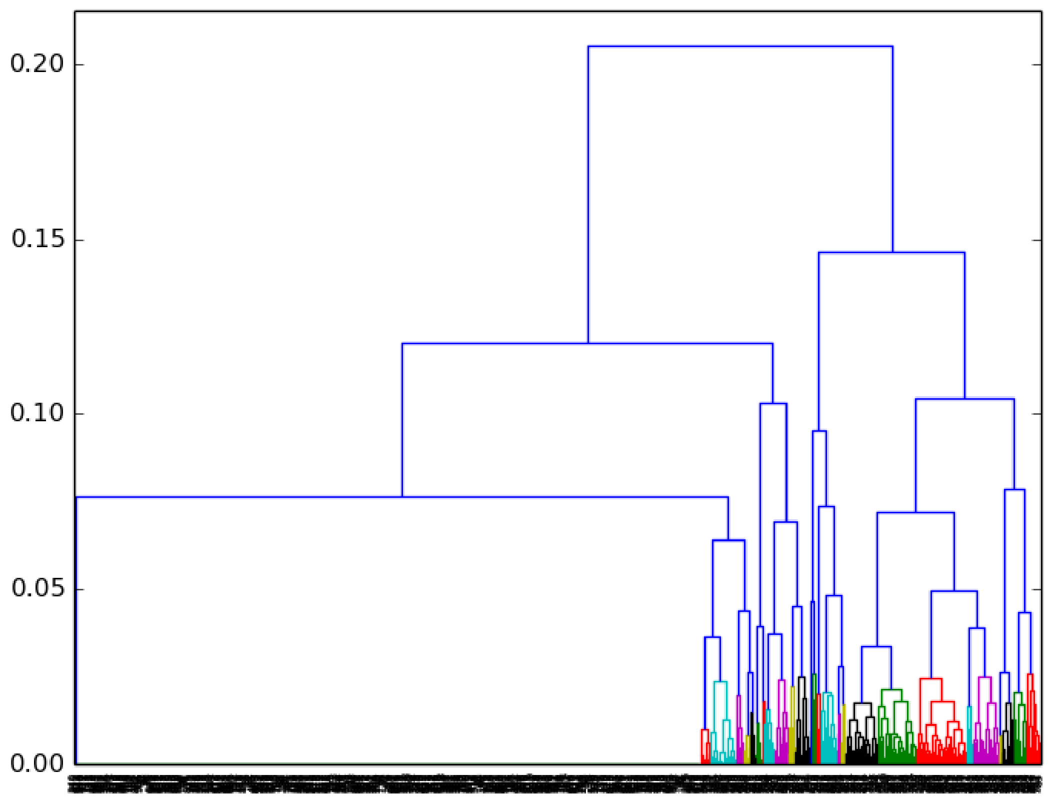
```
#sort result from smallest distance to largest
userSimilarityMatrix = calculateSimilarItems
(
    prefs=prefs,
    simMeasure=sim_distance,
    n=5,
    reverseSimilarityFlag=False,
    transformMatrixFlag=False
)
The result of this was written to hierarchicalClusterInput1.txt
```

   (b) **Search for hierarchical cluster with 5 items:** This was achieved through Listing 7. By iteratively and incrementally generating hierarchical clusters. The search resulted in a clustering of 27 containing 5 raters: (551, 789, 814, 846, 881). Figure 1. outlines the dendogram.

Listing 7: Hierarchical Clustering with Dendogram Plot

```
#cluster and plot dendogram
def plotHierarchicalClustering(items2dArray, maxClusterCount, labelArray,
    searchItemCount):
```

Figure 1: Hierarchical Cluster with a Cluster of 5 Closest Raters (551, 789, 814, 846, 881)



(a) Hierarchical Clustering

Problem 1 continued on next page. . .

```
    if( len(items2dArray) > 0 and len(items2dArray) >= maxClusterCount and
        searchItemCount > 0):

        distances = hierarchicalClustering.distance.pdist(items2dArray)

        linkages = hierarchicalClustering.linkage(distances, method='
            complete')
        clusterData = hierarchicalClustering.fcluster(linkages,
            maxClusterCount, 'maxclust')

        clusterClusterItemsDict = {}

        # calculate labels and count the number of items in each cluster
        #labels=list('' for i in range(len(items2dArray)))
        for i in range(len(items2dArray)):

            #labels[i]=str(i)+ ',' + str(clusterData[i])
            #print clusterData[i]
            clusterIndex = clusterData[i]
            clusterClusterItemsDict.setdefault(clusterIndex, [])
            clusterClusterItemsDict[clusterIndex].append(i)

        # calculate color threshold
        ct=linkages[-(maxClusterCount-1),2]

        #print clusterClusterCountDict
        showPlotFlag = False
        for clusterIndex, clusterItemsArray in clusterClusterItemsDict.items
            ():
            if( searchItemCount == len(clusterItemsArray) ):
                print 'yes, maxClusterCount: ', maxClusterCount
                print clusterClusterItemsDict[clusterIndex]
                showPlotFlag = True
                break

        if( showPlotFlag ):
            #plot
            P =hierarchicalClustering.dendrogram(linkages, labels =
                labelArray, color_threshold = ct)
            plt.show()
```

In order to get the average Pearson's r (sorted by r), Listing 8. computes pairwise Pearson similarity matrix for the 5 raters and calcuates the averages of each row, with the result outlined in Table 8.

Listing 8: Pearson's similarity for 5 raters

```
#average similarity sorted by r
arrayOfItems = [551, 789, 814, 846, 881]
dictionaryOfPearsonSimilarity = {}

#pairwise pearson matrix
for item in arrayOfItems:
    dictionaryOfPearsonSimilarity[item] = [ sim_pearson(prefs, str(entry), str
        (item)) for entry in arrayOfItems if entry!=item]
```

```

raterAveragePearsonArrayOfTuples = []
for rater, pairwiseArray in dictionaryOfPearsonSimilarity.items():
    raterDataTuple = (rater, sum(pairwiseArray)/float(len(pairwiseArray)))
    raterAveragePearsonArrayOfTuples.append(raterDataTuple)
#sort
raterAveragePearsonArrayOfTuples = sorted(raterAveragePearsonArrayOfTuples,
    key=lambda tup: tup[1], reverse=True)
print raterAveragePearsonArrayOfTuples
```

Table 8: Pearson's Coefficient for 5 Similar Raters

ITEM	RATER	CORRELATION COEFFICIENT
1	789	0.3316
2	881	0.2847
3	846	0.2545
4	551	0.0740
5	814	0.0566

8. **Solution 8:** Solution 8 follows the same pathway (distance matrix, search clusterings) as solution 7 with the exception of the call:

```
#sort result from largest distance to smallest
userSimilarityMatrix = calculateSimilarItems
(
    prefs=prefs,
    simMeasure=sim_distance,
    n=5,
    reverseSimilarityFlag=True,
    transformMatrixFlag=False
)

The result of this was written to hierarchicalClusterInput2.txt
```

The 5 raters who least agree are: (193, 206, 233, 302, 825). Table 9. was achieved in a similar fashion as solution 7.

Table 9: Pearson's Coefficient for 5 least Similar Raters

ITEM	RATER	CORRELATION COEFFICIENT
1	825	-0.2781
2	233	-0.0989
3	206	-0.0606
4	302	0.1550
5	193	0.2395

9. **Solution 9:** By applying an age filter on the aggregate male data and using the result to query the aggregate movie data, Listing 9. gets the highest rated movies by men above/below 40 (Table 10 and 11.)

Listing 9: Highest Rated Movies By Men/Women Above/Below age

```
#output:
    #aggregateUsersFemale: [ (user id, F, {'movie_title': movie rating}) ]
def getHighestRatedMoviesByMenOrWomenOverAge(aggregateUsersFemaleOrMale, count,
    ageLimit, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and
         len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []

        #aggregate ratings per movie
        for user in aggregateUsersFemaleOrMale:

            #if this person is beyond ageLimit
            if( int(user[2]) > ageLimit ):
                #userId: user[0]
                #gender: user[1]
                #Age: user[2]
                #<movie title,rating>: user[2]
                #tupleOfMovieRatingDictionary[]
                for movie, rating in user[3].items():

                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)


        #average ratings
        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)

        #sort
        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1

#output:
    #aggregateUsersFemale: [ (user id, F, {'movie_title': movie rating}) ]
def getHighestRatedMoviesByMenOrWomenUnderAge(aggregateUsersFemaleOrMale, count,
    ageLimit, movies):
```

Problem 1 continued on next page. . .

```
    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and
         len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []

        #aggregate ratings per movie
        for user in aggregateUsersFemaleOrMale:

            #if this person is beyond ageLimit
            if( int(user[2]) < ageLimit ):
                #userId: user[0]
                #gender: user[1]
                #Age: user[2]
                #<movie title,rating>: user[2]
                #tupleOfMovieRatingDictionary[]
                for movie, rating in user[3].items():

                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)

        #average ratings
        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)

        #sort
        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1
```

Table 10: Arbitrary Subset of Highest Rated Movies by Men over 40

ITEM	MOVIE-NAME	RATING
1	Great Day in Harlem, A (1994)	5.0
2	Two or Three Things I Know About Her (1966)	5.0
3	Aparajito (1956)	5.0
4	Strawberry and Chocolate (Fresa y chocolate) (1993)	5.0
5	Little Princess, The (1939)	5.0

Problem 1 continued on next page. . .

Table 11: Arbitrary Subset of Highest Rated Movies by Men under 40

ITEM	MOVIE-NAME	RATING
1	Entertaining Angels: The Dorothy Day Story (1996)	5.0
2	Letter From Death Row, A (1998)	5.0
3	Hugo Pool (1997)	5.0
4	Leading Man, The (1996)	5.0
5	Quiet Room, The (1996)	5.0

10. **Solution 10:** Solution 10 uses the same means as solution 9 to get the result highest rated movies by women above/below 40 with the exception of passing the getHighestRatedMoviesByMenOrWomen-OverAge() (Listing 9.) and getHighestRatedMoviesByMenOrWomenUnderAge() (Listing 9.) the aggregate female data. Table 12 and 13. outlines the results.

Table 12: Arbitrary Subset of Highest Rated Movies by Women over 40

ITEM	MOVIE-NAME	RATING
1	In the Bleak Midwinter (1995)	5.0
2	Foreign Correspondent (1940)	5.0
3	Swept from the Sea (1997)	5.0
4	Great Dictator, The (1940)	5.0
5	Balto (1995) The (1939)	5.0

Table 13: Arbitrary Subset of Highest Rated Movies by Women under 40

ITEM	MOVIE-NAME	RATING
1	Nico Icon (1995)	5.0
2	Backbeat (1993)	5.0
3	Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964)	5.0
4	Everest (1998)	5.0
5	Someone Else's America (1995)	5.0