Assignment Eight

Sybil Melton

November 13, 2014

CONTENTS

LISTINGS

1 MovieLens

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
2. u.item
3. u.user

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 10 points. You must have the question answered completely correct; partial credit will only be awarded if your answer is very close to the correct one.

1.1 Highest Average

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

1.1.1 Solution

The first step was to load the data into a usable structure. The movie title to id mappings were used frequently, so getMovieTitles was written from the loadMovieLens function, to be reusable and returns a dictionary of the mappings. Module loadMovieRatings is a modified version of loadMovieLens. It was written to take the user ratings from u.data, combine it with the movie titles, and return a dictionary of the movie to user rating mappings with the 'ratings' key. [4] This key was used to keep track of the averages later on. Since there were several requirements for rating averages, ratingAverage was developed for calculating and returning the average of a list of ratings, based on the 'ratings' key. There were also several instances of printing top rating averages, so printRatings was written to print each result. The To cover the event that there was a tie of more than five, getTop allowed for printing all of the items that have the top result. If that list was less than five results, it continued to look to fill the requirement. The module getRatings combined all of these functions. The movie to user rating mappings were loaded into a dictionary, then for each movie, the average was added to the dictionary with the 'average' key. The dictionary was turned into a list of (key, value) tuples, and sent to printRatings. [1] The formatting used left and right justification to get an easier to read output. [3] Listing 1 is the functions used.

```
1  def getMovieTitles(path='data'):
2    # Get movie titles
3      movies = {}
4      for line in open(path + '/u.item'):
5          (id, title) = line.split('|')[0:2]
6          movies[id] = title
7      return movies
8
9  def loadMovieRatings(path='data'):  #load movies with title and ratings list
10     # Get movie titles
11     movies = getMovieTitles()
12     # Get all ratings for each movie
```

```
13      rate = {}
14      for line in open(path + '/u.data'):
15          (user, movieid, rating, ts) = line.split('\t')
16          rate.setdefault(movieid, {})
17          rate[movieid]['title'] = movies[movieid]
18          if rate[movieid].has_key('ratings'):
19              rate[movieid]['ratings'].append(float(rating))
20          else:
21              rate[movieid]['ratings'] = []
22              rate[movieid]['ratings'].append(float(rating))
23      return rate
24
25  def ratingAverage(ratings):  #get average of a list of movie ratings
26      sum = 0.0
27      for i in range(0, len(ratings['ratings'])):
28          sum = sum + ratings['ratings'][i]
29      avg = sum / len(ratings['ratings'])
30      return avg
31
32  def getTop(sort, type, n=5):
33      max = sort[0][1][type]
34      top = []
35      top.append(sort[0])
36      for s in range(1, len(sort)):
37          if sort[s][1][type] == max:
38              top.append(sort[s])
39          else:
40              break
41      while len(top) < n:
42          max = sort[len(top)][1][type]
43          for s in range(len(top), len(sort)):
44              if sort[s][1][type] == max:
45                  top.append(sort[s])
46              else:
47                  break
48      return top
49
50  def printRatings(items, type, n):  #print list of ratings
51      sort = sorted(items, key=lambda x: x[1][type], reverse=True)
52      col_width = 0
53      top = getTop(sort, type, n)
54      # Get longest title for printed column width
55      for x in range(0, len(top)):
56          if len(top[x][1]['title']) > col_width:
57              col_width = len(top[x][1]['title']) + 2
58      # print the result
59      print string.ljust('Title', col_width), string.rjust('Average Rating', 3)
60      print string.ljust('\n------', col_width), string.rjust('----------------', 3)
61      for x in range(0, len(top)):
62          print string.ljust(top[x][1]['title'], col_width), string.rjust(str(top[x][1][type]), 3)
63
64  def getRatings(n=5):  #get top average, most rated
65      rate = loadMovieRatings()
66      # Get the average ratings per movie
67      for r in rate:
68          rate[r]['average'] = ratingAverage(rate[r])
69      print 'Top ' + str(n) + ' movies by Average Rating:'
70      printRatings(rate.items(), 'average', n)
```

Listing 1: Get Highest Average Modules

1.1.2 Result

The top movies all had a rating of 5.0. In my testing I chose a number higher than 5, to show that my functions were working correctly. The results were saved into results.txt and are included in this report. There were more than five, but for the purposes of the assignment, the output still states "Top 5." Figure 1.1 shows the list.

```
Top 5 movies by Average Rating:
Title                                              Average Rating
------                                             ----------------
Entertaining Angels: The Dorothy Day Story (1996)   5.0
Star Kid (1997)                                    5.0
Great Day in Harlem, A (1994)                      5.0
They Made Me a Criminal (1939)                     5.0
Someone Else's America (1995)                      5.0
Saint of Fort Washington, The (1993)               5.0
Aiqing wansui (1994)                               5.0
Santa with Muscles (1996)                          5.0
Prefontaine (1997)                                 5.0
Marlene Dietrich: Shadow and Light (1996)          5.0
```

Figure 1.1: Highest Average Ratings

1.2 Most Ratings

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

1.2.1 Solution

The same function, getRatings, was used as inthe first question. The data was sorted differently, on the length of the ratings list, for each movie. Since the sorting had to be completed by the rating list length, instead of an average value , the print statements had to be written separately. Listing 2 is the additional code used in the getRatings function.

```
1     # Get the most rated movies
2     rsort = sorted(rate.items(), key=lambda x: len(x[1]['ratings']), reverse=True)
3     print '\nTop ' + str(n) + ' movies by most ratings:'
4     col_width = 0
5     for x in range(0, n):
6         if len(rsort[x][1]['title']) > col_width:
7             col_width = len(rsort[x][1]['title']) + 2
8     print string.ljust('Title', col_width), string.rjust('Number of Ratings', 3)
9     print string.ljust('------', col_width), string.rjust('----------------', 3)
10    for x in range(0, n):
11        print string.ljust(rsort[x][1]['title'], col_width), string.rjust(str(len(rsort[x][1]['ratings'])),
              3)
```

Listing 2: Get Most Ratings Code

1.2.2 Result

There were no surprises to me, in this list of top rated movies, since I have seen all of them. I did not see any ties, so for the assignment, it was not included. Figure 1.2 shows the output.

```
Top 5 movies by most ratings:
Title                          Number of Ratings
------                         ----------------
Star Wars (1977)               583
Contact (1997)                 509
Fargo (1996)                   508
Return of the Jedi (1983)      507
Liar Liar (1997)               485
```

Figure 1.2: Most Ratings

1.3 Highest Average by Women

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

1.3.1 Solution

A separate module was written for the gender classifications, because additional user information was needed. Module loadUserInfo placed the user id to movie rating mappings with a nested dictionary with age and gender. [1] The loadMovieRatings module was used again, for consistency and reuse of average and printing functions. A dictionary was created for women, initialized with each movie id. If a user's gender was F, for female, the movie list was checked if the user had rated it and the rating was added to the women dictionary. Once all the women ratings were found, the average for each was calculated using ratingAverage and saved in the dictionary with movies. In order to prevent KeyErrors, if there was a movie with no average for women, it was set to negative infinity. [2] Since we were only interested in the highest ratings, it had no affect on the results. Then the results were printed using printRatings, as shown in Listing 1, for the top highest average. Listing 3 is the loadUserInfo and getRatingsByGender modules.

```
1  def loadUserInfo(path='data'):  #load users with info and movie ratings
2    # Load data
3      prefs = {}
4      for line in open(path + '/u.data'):
5          (user, movieid, rating, ts) = line.split('\t')
6          prefs.setdefault(user, {})
7          prefs[user][movieid] = float(rating)
8      for line in open(path + '/u.user'):
9          (user, age, gender, job, zip) = line.split('|')
10         prefs[user]['info'] = {}
11         prefs[user]['info']['age'] = int(age)
12         prefs[user]['info']['gender'] = gender
13     return prefs
14
15 def getRatingsByGender(n=5):  #get top average, most rated, ratings by gender
16     rate = loadMovieRatings()
17         # Get ratings per gender
18     users = loadUserInfo()
19     women = {}
20     men = {}
21     for r in rate:
22         women[r] = {}
23         men[r] = {}
24     for u in users:
25         if users[u]['info']['gender'] == 'F':
26             for r in rate:
27                 if users[u].has_key(r):
28                     if women[r].has_key('ratings'):
29                         women[r]['ratings'].append(users[u][r])
30                     else:
```

```
                        women[r]['ratings'] = []
                        women[r]['ratings'].append(users[u][r])
        else:
            for r in rate:
                if users[u].has_key(r):
                    if men[r].has_key('ratings'):
                        men[r]['ratings'].append(users[u][r])
                    else:
                        men[r]['ratings'] = []
                        men[r]['ratings'].append(users[u][r])
    # Get the average ratings per movie per gender
    for w in women:
        if women[w].has_key('ratings'):
            rate[w]['women'] = ratingAverage(women[w])
    for m in men:
        if men[m].has_key('ratings'):
            rate[m]['men'] = ratingAverage(men[m])
    for r in rate:
        if rate[r].has_key('women'):
            continue
        else:
            rate[r]['women'] = 0
    for r in rate:
        if rate[r].has_key('men'):
            continue
        else:
            rate[r]['men'] = 0
    # Print results per gender
    print '\nTop ' + str(n) + ' movies Rated by Women:'
    printRatings(rate.items(), 'women', n)
    print '\nTop ' + str(n) + ' movies Rated by Men:'
    printRatings(rate.items(), 'men', n)
```

Listing 3: Get Top Ratings by Gender Module

1.3.2 Result

Surprisingly, only one movie, Prefontaine (1997), was in both highest average list and the women's top ratings list. This leads me to believe that men and women tend to disagree on movie ratings. There were also more then five, but the output says "Top 5" to meet the question requirement. Figure 1.3 shows the list.

```
Top 5 movies Rated by Women:
Title                                 Average Rating
------                                ----------------
Someone Else's America (1995)          5.0
Foreign Correspondent (1940)           5.0
Visitors, The (Visiteurs, Les) (1993)  5.0
Stripes (1981)                         5.0
Telling Lies in America (1997)         5.0
Year of the Horse (1997)               5.0
Faster Pussycat! Kill! Kill! (1965)    5.0
Everest (1998)                         5.0
Maya Lin: A Strong Clear Vision (1994) 5.0
Prefontaine (1997)                     5.0
Mina Tannenbaum (1994)                 5.0
```

Figure 1.3: Highest Average Ratings by Women

1.4 Highest Average by Men

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

1.4.1 Solution

The same module was used for men as it was for women. A separate dictionary was created for men, to store the movie ratings and average. If a user's gender was not F, the movie list was checked if the user had rated it and if so, the rating was added to the men dictionary. Once all the men ratings were found, the average for each was calculated using ratingAverage and saved in the dictionary with movies. In order to prevent KeyErrors, if no man rated a movie, it was set to negative infinity, as with the code for women. [2] The code for men was included in listing 3.

1.4.2 Result

I found that there were more top rated movies by men that matched the highest average movies from question 1. This tells me that women must not have rated the top rated movies by men that also appear on the highest average list. Prefontaine (1997) was on the list for both men and women, so everyone rated that movie equally high. Again, there were more than five, but for the purposes of the assignment, the output still states "Top 5." Figure 1.4 shows the output.

```
Top 5 movies Rated by Men:
Title                                          Average Rating
------                                        ----------------
Entertaining Angels: The Dorothy Day Story (1996)   5.0
Letter From Death Row, A (1998)                     5.0
Hugo Pool (1997)                                    5.0
Leading Man, The (1996)                             5.0
Quiet Room, The (1996)                              5.0
Love Serenade (1996)                                5.0
Star Kid (1997)                                     5.0
Great Day in Harlem, A (1994)                       5.0
They Made Me a Criminal (1939)                      5.0
Delta of Venus (1994)                               5.0
Saint of Fort Washington, The (1993)                5.0
Aiqing wansui (1994)                                5.0
Little City (1998)                                  5.0
Santa with Muscles (1996)                           5.0
Prefontaine (1997)                                  5.0
Marlene Dietrich: Shadow and Light (1996)           5.0
```

Figure 1.4: Highest Average Ratings by Men

1.5 Top Gun Correlation

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

1.5.1 Solution

In order to find a movie with the closest correlation to Top Gun, calculateSimilarItems, which was included in the original code, was used. The original was written only for movies, so it was modified to allow for finding similar users as well. The statement to invert the dictionary was removed and I made sure I was sending the correct dictionary to be used. Additionally, it was set to use the sim_pearson function, for correlation. To get the lowest negative correlation, two additional functions were written similarly. The main difference was botMatches was written to return the worst correlation. The modules for sim_pearson was unchanged. The loadMovie was used to retrieve with the user rating list, per movie, which is the reverse of the original loadMovieLens. This allowed for the correlation to be calculated based on user ratings per movie. [4] Top Gun had to be found in the movie list, so the movie most similar and dissimilar could be retrieved from the results of calculateSimilarItems and

calculateDissimilarItems. Listing 4 is the functions written to get the Top Gun correlation.

```
def topMatches(prefs, original, n=5, similarity=sim_pearson):
    '''
    Returns the best matches for an item from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, original, other), other) for other in prefs
              if other != original]
    scores.sort()
    scores.reverse()
    return scores[0:n]

def botMatches(prefs, original, n=5, similarity=sim_pearson):
    '''
    Returns the worst matches for an item from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, original, other), other) for other in prefs
              if other != original]
    scores.sort()
    return scores[0:n]

def calculateSimilarItems(prefs, n=10):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''
    result = {}
    c = 0
    for item in prefs:
        # Status updates for large datasets
        c += 1
        if c % 100 == 0:
            print '%d / %d' % (c, len(prefs))
        # Find the most similar items to this one
        scores = topMatches(prefs, item, n=n, similarity=sim_distance)
        result[item] = scores
    return result

def calculateDissimilarItems(prefs, n=10):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''
    result = {}
    c = 0
    for item in prefs:
        # Status updates for large datasets
        c += 1
        if c % 100 == 0:
            print '%d / %d' % (c, len(prefs))
        # Find the most similar items to this one
        scores = botMatches(prefs, item, n, similarity=sim_distance)
        result[item] = scores
    return result

def loadMovie(path='data'): #movies, with per user ratings
  # Get movie titles
    movies = getMovieTitles()
  # Load data
    prefs = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(movieid, {})
        prefs[movieid][user] = float(rating)
    return prefs

def getTopGunCorrelation(path='data'): #get movies most alike and unlike Top Gun
    #load list of movies, with per user ratings
    mList = loadMovie()
    topGun = 0
```

```
71      # Get movie titles
72      movies = getMovieTitles()
73      for m in movies:
74          if 'Top Gun' in movies[m]:
75              topGun = m
76      sim = calculateSimilarItems(mList, 1)
77      neg = calculateDissimilarItems(mList, 1)
78      print '\nMovie most like Top Gun: '
79      print movies[sim[m][0][1]]
80      print '\nMovie least like Top Gun: '
81      print movies[neg[m][0][1]]
```

Listing 4: Top Gun Correlation Module

1.5.2 Result

Figure 1.5 shows the movies with ratings most like and most unlike Top Gun. I have not seen either of the movies in the output, so I do not have an opinion about the correlation.

```
Movie most like Top Gun:
Last Time I Committed Suicide, The (1997)

Movie least like Top Gun:
Lightning Jack (1994)
```

Figure 1.5: Top Gun Correlation

1.6 Top Raters

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

1.6.1 Solution

A new module, loadUserRatings, was written to compile a dictionary of users with a list of their movie ratings. Using this, the top raters were found by using the length of the list of movie ratings. Listing 5 shows the loadUserRatings and getTopRaters module.

```
1  def loadUserRatings(path='data'): # users with per movie ratings
2    # Get movie titles
3      movies = getMovieTitles()
4    # Load data
5      prefs = {}
6      for line in open(path + '/u.data'):
7          (user, movieid, rating, ts) = line.split('\t')
8          prefs.setdefault(user, {})
9          prefs[user][movieid] = float(rating)
10     return prefs
11
12 def getTopRaters(n=5):  #get users that rated the most movies
13     users = loadUserRatings()
14     rsort = sorted(users.items(), key=lambda x: len(x[1]), reverse=True)
15     print '\nTop 5 users by most ratings:'
16     print string.ljust('User', 12), string.rjust('Number of Ratings', 3)
17     print string.ljust('------', 12), string.rjust('----------------', 3)
18     for x in range(0, n):
19         print string.ljust(rsort[x][0], 12), string.rjust(str(len(rsort[x][1])), 3)
```

Listing 5: Top Raters Module

1.6.2 Result

There were no ties, so the top five were printed. Figure 1.6 shows the user ID and number of films each rated.

```
Top 5 users by most ratings:
User          Number of Ratings
------        ----------------
405           737
655           685
13            636
450           540
276           518
```

Figure 1.6: Top Raters

1.7 User Agreement

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

1.7.1 Solution

My solution used calculateSimilarItems to find the user closest to each. For each user, a chain was created by joining the user closest to it. Each linked userid was put into a list and the links with r and distance values were put together into a dictionary. The list of links were converted to a set, to keep the values unique, because in testing I found some users linked back to each other, creating a loop. At the end of the assignment I realized I could have grabbed more results and added to the chain with the next closest user, but there was not enough time to test it. The chain was searched for the shortest distance along the chain. Listing 1.7.1 is the module.

```
1  def getUserCorrelation(n=5):  #get users that are most correlated with each other
2      '''
3      Finds the 5 users that agree the most by finding the users that are closest
4      '''
5      users = loadUserRatings()
6      alike = calculateSimilarItems(users, 1)
7      min = float('inf')
8      minid = 0
9      chain = {}
10     r = {}
11     for a in alike:
12         sum = 0
13         c = alike[a]
14         chain[a] = {}
15         links = []
16         links.append(a)
17         r[a] = [a, alike[a][0][1], sim_pearson(users, str(a), alike[a][0][1]), sim_distance(users, str(a),
               alike[a][0][1])]
18         sum = sum + sim_distance(users, str(a), alike[a][0][1])
19         for i in range(0, n):
20             add = links[i]
21             links.append(alike[add][0][1])
22             next = alike[alike[add][0][1]][0][1]
23             sum = sum + sim_distance(users, alike[add][0][1], alike[alike[add][0][1]][0][1])
```

```
            chain[a]['links'] = list(set(links))
        chain[a]['distance'] = sum
    for c in chain:
        if len(chain[c]['links']) >= 5:
            if chain[c]['distance'] < min:
                min = chain[c]['distance']
                minid = c
    chain[minid]['r'] = {}
    for i in range(0, len(chain[minid]['links'])):
        chain[minid]['r'][i] = r[chain[minid]['links'][i]]
    csort = sorted(chain[minid]['r'].items(), key=lambda x: x[1][2], reverse=True)
    print '5 closest raters with r values and distance: '
    for c in csort:
        print 'User ' + c[1][0] + ' to ' + c[1][1] + ', r = ' + str(c[1][2]) + ', distance= ' + str(c
            [1][3])
```

1.7.2 Result

I would think that if two users had ratings close to each other, the r value would be positively correlated. My results did not support this, the Pearson value was calculated to be zero. When testing, this intuition seemed to be correct for chains up to four. This leads me to believe that my algorithm for finding the five closest was not correct. Figure 1.7 shows the result.

```
5 closest raters with r values and distance:
User 796 to 341, r = 0, distance= 0.2
User 908 to 88, r = 0, distance= 1.0
User 95 to 88, r = 0, distance= 1.0
User 341 to 908, r = 0, distance= 1.0
User 88 to 95, r = 0, distance= 1.0
```

Figure 1.7: User Agreement

1.8 User Disagreement

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

1.8.1 Solution

The code to find the 5 raters who disagreed was the same as the code for agreement, except using calcuateDissimilarItems. Listing 6 is the additional code in getUserCorrelation.

```
    #Finds the 5 users that disagree the most by finding the users that are farthest away
    unlike = calculateDissimilarItems(users, 1)
    max = float('-inf')
    maxid = 0
    uchain = {}
    ur = {}
    for u in unlike:
        sum = 0
        c = unlike[u]
        uchain[u] = {}
        links = []
        links.append(u)
        ur[u] = [a, unlike[u][0][1], sim_pearson(users, str(u), unlike[u][0][1]), sim_distance(users, str(u
            ), unlike[u][0][1])]
        sum = sum + sim_distance(users, str(u), unlike[u][0][1])
        for i in range(0, n):
            add = links[i]
```

```
            links.append(unlike[add][0][1])
            next = unlike[unlike[add][0][1]][0][1]
            sum = sum + sim_distance(users, unlike[add][0][1], unlike[unlike[add][0][1]][0][1])
            uchain[u]['links'] = list(set(links))
        uchain[u]['distance'] = sum
    for c in uchain:
        if len(uchain[c]['links']) == 5:
            if uchain[c]['distance'] > max:
                max = uchain[c]['distance']
                maxid = c
    uchain[maxid]['ur'] = {}
    for i in range(0, len(uchain[maxid]['links'])):
        uchain[maxid]['ur'][i] = ur[uchain[maxid]['links'][i]]
    csort = sorted(uchain[maxid]['ur'].items(), key=lambda x: x[1][2], reverse=True)
    print '5 farthest raters, with r values and distance: '
    for c in csort:
        print 'User ' + c[1][0] + ' to ' + c[1][1] + ', r = ' + str(c[1][2]) + ', distance= ' + str(c
            [1][3])
```

Listing 6: User Disagreement

1.8.2 Result

My results for disagreement also do not follow what I would expect. All of the Pearson r values were zero, whereas I would expect them to be negative. Figure **??** shows the result.

```
5 farthest raters, with r values and distance:
User 479 to 445, r = 0.15494918816, distance= 0.00636942675159
User 479 to 124, r = 0, distance= 0
User 479 to 100, r = 0, distance= 0
User 479 to 100, r = 0, distance= 0
User 479 to 565, r = 0, distance= 0
```

Figure 1.8: User Disagreement

1.9 Men by Age

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

1.9.1 Solution

This solution was similar to the gender based solution. The modules used to import the data were loadMovieRatings from listing 1 and loadUserInfo from listing 3. Dictionaries were created for both gender based age groups, so the averages could be calculated independently. Then for each user, gender and age were checked, and the rating for each movie was added. Next the averages were calculated using ratingAverage, as in listing 1. If the movie was not rated by that group, it was set to negative infinity, since it would not affect the highest average outcome. Then the results were printed using printRatings, also from listing 1. Listing 7 is the function getRatingsByAge.

```
def getRatingsByAge(n=5):  #get top average, most rated, ratings by gender
    rate = loadMovieRatings()
    users = loadUserInfo()
    women = {}
    w40 = {}
    men = {}
    m40 = {}
    for r in rate:
        women[r] = {}
        men[r] = {}
        w40[r] = {}
```

```
12          m40[r] = {}
13      for u in users:
14          if users[u]['info']['gender'] == 'F':
15              if int(users[u]['info']['age']) < 40:
16                  for r in rate:
17                      if users[u].has_key(r):
18                          if women[r].has_key('ratings'):
19                              women[r]['ratings'].append(users[u][r])
20                          else:
21                              women[r]['ratings'] = []
22                              women[r]['ratings'].append(users[u][r])
23              else:
24                  for r in rate:
25                      if users[u].has_key(r):
26                          if w40[r].has_key('ratings'):
27                              w40[r]['ratings'].append(users[u][r])
28                          else:
29                              w40[r]['ratings'] = []
30                              w40[r]['ratings'].append(users[u][r])
31          else:
32              if int(users[u]['info']['age']) < 40:
33                  for r in rate:
34                      if users[u].has_key(r):
35                          if men[r].has_key('ratings'):
36                              men[r]['ratings'].append(users[u][r])
37                          else:
38                              men[r]['ratings'] = []
39                              men[r]['ratings'].append(users[u][r])
40              else:
41                  for r in rate:
42                      if users[u].has_key(r):
43                          if m40[r].has_key('ratings'):
44                              m40[r]['ratings'].append(users[u][r])
45                          else:
46                              m40[r]['ratings'] = []
47                              m40[r]['ratings'].append(users[u][r])
48      # Get the average ratings per movie per gender
49      for w in women:
50          if women[w].has_key('ratings'):
51              rate[w]['women'] = ratingAverage(women[w])
52          else:
53              rate[w]['women'] = float('-inf')
54      for m in men:
55          if men[m].has_key('ratings'):
56              rate[m]['men'] = ratingAverage(men[m])
57          else:
58              rate[m]['men'] = float('-inf')
59      for w in w40:
60          if w40[w].has_key('ratings'):
61              rate[w]['w40'] = ratingAverage(w40[w])
62          else:
63              rate[w]['w40'] = float('-inf')
64      for m in m40:
65          if m40[m].has_key('ratings'):
66              rate[m]['m40'] = ratingAverage(m40[m])
67          else:
68              rate[m]['m40'] = float('-inf')
69      # Print results per gender
70      print '\nTop ' + str(n) + ' movies Rated by Men under 40:'
71      printRatings(rate.items(), 'men', n)
72      print '\nTop ' + str(n) + ' movies Rated by Men over 40:'
73      printRatings(rate.items(), 'm40', n)
74      print '\nTop ' + str(n) + ' movies Rated by Women under 40:'
75      printRatings(rate.items(), 'women', n)
76      print '\nTop ' + str(n) + ' movies Rated by Women over 40:'
77      printRatings(rate.items(), 'w40', n)
```

Listing 7: Ratings by Age

1.9.2 Result

Althought the requirement was for five, there were more that had a average rating of 5.0. There were a few movies from the top rated men's list that were also on the list for men over 40. Figure 1.9 shows the list. However, most of the movies on the men under 40 list, as shown in Figure 1.10, were also on the top rated men's list. This tells me that most of the movies on the top rated men's list were not rated by men over 40. The movie Prefontaine (1997) was on both lists, so again, it was liked by everyone.

```
Top 5 movies Rated by Men over 40:
Title                                                 Average Rating
------                                               ----------------
Leading Man, The (1996)                               5.0
Faithful (1996)                                       5.0
Rendezvous in Paris (Rendez-vous de Paris, Les) (1995)  5.0
Aparajito (1956)                                      5.0
Star Kid (1997)                                       5.0
Great Day in Harlem, A (1994)                         5.0
They Made Me a Criminal (1939)                        5.0
Poison Ivy II (1995)                                  5.0
Two or Three Things I Know About Her (1966)           5.0
Little Princess, The (1939)                           5.0
Late Bloomers (1996)                                  5.0
Solo (1996)                                           5.0
Grateful Dead (1995)                                  5.0
Hearts and Minds (1996)                               5.0
Little City (1998)                                    5.0
Boxing Helena (1993)                                  5.0
World of Apu, The (Apur Sansar) (1959)                5.0
Spice World (1997)                                    5.0
Double Happiness (1994)                               5.0
Bitter Sugar (Azucar Amargo) (1996)                   5.0
Prefontaine (1997)                                    5.0
Marlene Dietrich: Shadow and Light (1996)             5.0
```

Figure 1.9: Top Ratings Men over 40

```
Top 5 movies Rated by Men under 40:
Title                                               Average Rating
------                                             ----------------
Entertaining Angels: The Dorothy Day Story (1996)   5.0
Letter From Death Row, A (1998)                     5.0
Hugo Pool (1997)                                    5.0
Leading Man, The (1996)                             5.0
Quiet Room, The (1996)                              5.0
Love Serenade (1996)                                5.0
Star Kid (1997)                                     5.0
Perfect Candidate, A (1996)                         5.0
Delta of Venus (1994)                               5.0
Love in the Afternoon (1957)                        5.0
Saint of Fort Washington, The (1993)                5.0
Aiqing wansui (1994)                                5.0
Crossfire (1947)                                    5.0
Santa with Muscles (1996)                           5.0
Magic Hour, The (1998)                              5.0
Angel Baby (1995)                                   5.0
Maya Lin: A Strong Clear Vision (1994)              5.0
Prefontaine (1997)                                  5.0
```

Figure 1.10: Top Ratings Men under 40

1.10

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

1.10.1 Solution

The solution for getting the ratings for women by age group was also in getRatingsByAge, shown in listing 7. Again, separate dictionaries were created to consolidate ratings and calculate the averages independently.

1.10.2 Result

There were few movies in the women over 40 group that were also in the women's top rated list. This was the first list that did not include the movie Prefontaine (1997), so women over 40 must not watch this movie. Figure **??** shows the list. As with the men, most of the movies in the women under 40 group were in the women's top rated list, shown in Figure **??**. Prefontaine (1997) was also in the list, as it was in both groups of men.

```
Top 5 movies Rated by Women over 40:
Title                                    Average Rating
------                                   ---------------
Nightmare Before Christmas, The (1993)   5.0
Letter From Death Row, A (1998)          5.0
Shall We Dance? (1937)                   5.0
Night Flier (1997)                       5.0
Pocahontas (1995)                        5.0
Wrong Trousers, The (1993)               5.0
Swept from the Sea (1997)                5.0
Great Dictator, The (1940)               5.0
Mrs. Winterbourne (1996)                 5.0
Safe (1995)                              5.0
Top Hat (1935)                           5.0
Foreign Correspondent (1940)             5.0
Ma vie en rose (My Life in Pink) (1997)  5.0
Visitors, The (Visiteurs, Les) (1993)    5.0
Grand Day Out, A (1992)                  5.0
Funny Face (1957)                        5.0
Bent (1997)                              5.0
Shallow Grave (1994)                     5.0
Tombstone (1993)                         5.0
In the Bleak Midwinter (1995)            5.0
U Turn (1997)                            5.0
Angel Baby (1995)                        5.0
Best Men (1997)                          5.0
Band Wagon, The (1953)                   5.0
Mina Tannenbaum (1994)                   5.0
```

Figure 1.11: Top Ratings Women over 40

Top 5 movies Rated by Women under 40:

Title	Average Rating
Nico Icon (1995)	5.0
Backbeat (1993)	5.0
Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964)	5.0
Someone Else's America (1995)	5.0
Don't Be a Menace to South Central While Drinking Your Juice in the Hood (1996)	5.0
Stripes (1981)	5.0
Heaven's Prisoners (1996)	5.0
Telling Lies in America (1997)	5.0
Year of the Horse (1997)	5.0
Faster Pussycat! Kill! Kill! (1965)	5.0
Everest (1998)	5.0
Grace of My Heart (1996)	5.0
Wedding Gift, The (1994)	5.0
Horseman on the Roof, The (Hussard sur le toit, Le) (1995)	5.0
Maya Lin: A Strong Clear Vision (1994)	5.0
Prefontaine (1997)	5.0
Mina Tannenbaum (1994)	5.0

Figure 1.12: Top Ratings Women under 40

1.11 Python Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import sqrt
import operator
import string

def sim_distance(prefs, p1, p2):
    '''
    Returns a distance-based similarity score for person1 and person2.
    '''
    # Get the list of shared_items
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    # If they have no ratings in common, return 0
    if len(si) == 0:
        return 0
    # Add up the squares of all the differences
    sum_of_squares = sum([pow(prefs[p1][item] - prefs[p2][item], 2) for item in
                          prefs[p1] if item in prefs[p2]])
    return 1 / (1 + sum_of_squares)

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 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:
```

```
72              # Similarity * Score
73              totals.setdefault(item, 0)
74              # The final score is calculated by multiplying each item by the
75              #    similarity and adding these products together
76              totals[item] += prefs[other][item] * sim
77              # Sum of similarities
78              simSums.setdefault(item, 0)
79              simSums[item] += sim
80      # Create the normalized list
81      rankings = [(total / simSums[item], item) for (item, total) in
82                  totals.items()]
83      # Return the sorted list
84      rankings.sort()
85      rankings.reverse()
86      return rankings
87
88  def transformPrefs(prefs):
89      '''
90      Transform the recommendations into a mapping where persons are described
91      with interest scores for a given title e.g. {title: person} instead of
92      {person: title}.
93      '''
94      result = {}
95      for person in prefs:
96          for item in prefs[person]:
97              result.setdefault(item, {})
98              # Flip item and person
99              result[item][person] = prefs[person][item]
100     return result
101
102 def getRecommendedItems(prefs, itemMatch, user):
103     userRatings = prefs[user]
104     scores = {}
105     totalSim = {}
106     # Loop over items rated by this user
107     for (item, rating) in userRatings.items():
108         # Loop over items similar to this one
109         for (similarity, item2) in itemMatch[item]:
110             # Ignore if this user has already rated this item
111             if item2 in userRatings:
112                 continue
113             # Weighted sum of rating times similarity
114             scores.setdefault(item2, 0)
115             scores[item2] += similarity * rating
116             # Sum of all the similarities
117             totalSim.setdefault(item2, 0)
118             totalSim[item2] += similarity
119     # Divide each total score by total weighting to get an average
120     rankings = [(score / totalSim[item], item) for (item, score) in
121                 scores.items()]
122     # Return the rankings from highest to lowest
123     rankings.sort()
124     rankings.reverse()
125     return rankings
126 #--------------------------------------------------------------------
127 def getMovieTitles(path='data'):
128   # Get movie titles
129     movies = {}
130     for line in open(path + '/u.item'):
131         (id, title) = line.split('|')[0:2]
132         movies[id] = title
133     return movies
134
135 def loadMovieRatings(path='data'):  #load movies with title and ratings list
136     # Get movie titles
137     movies = getMovieTitles()
138     # Get all ratings for each movie
139     rate = {}
140     for line in open(path + '/u.data'):
141         (user, movieid, rating, ts) = line.split('\t')
142         rate.setdefault(movieid, {})
143         rate[movieid]['title'] = movies[movieid]
144         if rate[movieid].has_key('ratings'):
```

```
145             rate[movieid]['ratings'].append(float(rating))
146         else:
147             rate[movieid]['ratings'] = []
148             rate[movieid]['ratings'].append(float(rating))
149     return rate
150
151 def ratingAverage(ratings):  #get average of a list of movie ratings
152     sum = 0.0
153     for i in range(0, len(ratings['ratings'])):
154         sum = sum + ratings['ratings'][i]
155     avg = sum / len(ratings['ratings'])
156     return avg
157
158 def getTop(sort, type, n=5):
159     max = sort[0][1][type]
160     top = []
161     top.append(sort[0])
162     for s in range(1, len(sort)):
163         if sort[s][1][type] == max:
164             top.append(sort[s])
165         else:
166             break
167     while len(top) < n:
168         max = sort[len(top)][1][type]
169         for s in range(len(top), len(sort)):
170             if sort[s][1][type] == max:
171                 top.append(sort[s])
172             else:
173                 break
174     return top
175
176 def printRatings(items, type, n):  #print list of ratings
177     sort = sorted(items, key=lambda x: x[1][type], reverse=True)
178     col_width = 0
179     top = getTop(sort, type, n)
180     # Get longest title for printed column width
181     for x in range(0, len(top)):
182         if len(top[x][1]['title']) > col_width:
183             col_width = len(top[x][1]['title']) + 2
184     # print the result
185     print string.ljust('Title', col_width), string.rjust('Average Rating', 3)
186     print string.ljust('\n------', col_width), string.rjust('----------------', 3)
187     for x in range(0, len(top)):
188         print string.ljust(top[x][1]['title'], col_width), string.rjust(str(top[x][1][type]), 3)
189
190 def getRatings(n=5):  #get top average, most rated
191     rate = loadMovieRatings()
192     # Get the average ratings per movie
193     for r in rate:
194         rate[r]['average'] = ratingAverage(rate[r])
195     print 'Top ' + str(n) + ' movies by Average Rating:'
196     printRatings(rate.items(), 'average', n)
197     # Get the most rated movies
198     rsort = sorted(rate.items(), key=lambda x: len(x[1]['ratings']), reverse=True)
199     print '\nTop ' + str(n) + ' movies by most ratings:'
200     col_width = 0
201     for x in range(0, n):
202         if len(rsort[x][1]['title']) > col_width:
203             col_width = len(rsort[x][1]['title']) + 2
204     print string.ljust('Title', col_width), string.rjust('Number of Ratings', 3)
205     print string.ljust('------', col_width), string.rjust('----------------', 3)
206     for x in range(0, n):
207         print string.ljust(rsort[x][1]['title'], col_width), string.rjust(str(len(rsort[x][1]['ratings'])),
              3)
208 #-----------------------------------------------------------------
209 def loadUserInfo(path='data'):  #load users with info and movie ratings
210   # Load data
211     prefs = {}
212     for line in open(path + '/u.data'):
213         (user, movieid, rating, ts) = line.split('\t')
214         prefs.setdefault(user, {})
215         prefs[user][movieid] = float(rating)
216     for line in open(path + '/u.user'):
```

```
217         (user, age, gender, job, zip) = line.split('|')
218         prefs[user]['info'] = {}
219         prefs[user]['info']['age'] = int(age)
220         prefs[user]['info']['gender'] = gender
221     return prefs
222 
223 def getRatingsByGender(n=5):  #get top average, most rated, ratings by gender
224     rate = loadMovieRatings()
225         # Get ratings per gender
226     users = loadUserInfo()
227     women = {}
228     men = {}
229     for r in rate:
230         women[r] = {}
231         men[r] = {}
232     for u in users:
233         if users[u]['info']['gender'] == 'F':
234             for r in rate:
235                 if users[u].has_key(r):
236                     if women[r].has_key('ratings'):
237                         women[r]['ratings'].append(users[u][r])
238                     else:
239                         women[r]['ratings'] = []
240                         women[r]['ratings'].append(users[u][r])
241         else:
242             for r in rate:
243                 if users[u].has_key(r):
244                     if men[r].has_key('ratings'):
245                         men[r]['ratings'].append(users[u][r])
246                     else:
247                         men[r]['ratings'] = []
248                         men[r]['ratings'].append(users[u][r])
249     # Get the average ratings per movie per gender
250     for w in women:
251         if women[w].has_key('ratings'):
252             rate[w]['women'] = ratingAverage(women[w])
253     for m in men:
254         if men[m].has_key('ratings'):
255             rate[m]['men'] = ratingAverage(men[m])
256     for r in rate:
257         if rate[r].has_key('women'):
258             continue
259         else:
260             rate[r]['women'] = 0
261     for r in rate:
262         if rate[r].has_key('men'):
263             continue
264         else:
265             rate[r]['men'] = 0
266     # Print results per gender
267     print '\nTop ' + str(n) + ' movies Rated by Women:'
268     printRatings(rate.items(), 'women', n)
269     print '\nTop ' + str(n) + ' movies Rated by Men:'
270     printRatings(rate.items(), 'men', n)
271 #-----------------------------------------------------------------
272 def topMatches(prefs, original, n=5, similarity=sim_pearson):
273     '''
274     Returns the best matches for an item from the prefs dictionary.
275     Number of results and similarity function are optional params.
276     '''
277     scores = [(similarity(prefs, original, other), other) for other in prefs
278               if other != original]
279     scores.sort()
280     scores.reverse()
281     return scores[0:n]
282 
283 def botMatches(prefs, original, n=5, similarity=sim_pearson):
284     '''
285     Returns the worst matches for an item from the prefs dictionary.
286     Number of results and similarity function are optional params.
287     '''
288     scores = [(similarity(prefs, original, other), other) for other in prefs
289               if other != original]
```

```
290     scores.sort()
291     return scores[0:n]
292
293 def calculateSimilarItems(prefs, n=10):
294     '''
295     Create a dictionary of items showing which other items they are
296     most similar to.
297     '''
298     result = {}
299     c = 0
300     for item in prefs:
301         # Status updates for large datasets
302         c += 1
303         if c % 100 == 0:
304             print '%d / %d' % (c, len(prefs))
305         # Find the most similar items to this one
306         scores = topMatches(prefs, item, n=n, similarity=sim_distance)
307         result[item] = scores
308     return result
309
310 def calculateDissimilarItems(prefs, n=10):
311     '''
312     Create a dictionary of items showing which other items they are
313     most similar to.
314     '''
315     result = {}
316     c = 0
317     for item in prefs:
318         # Status updates for large datasets
319         c += 1
320         if c % 100 == 0:
321             print '%d / %d' % (c, len(prefs))
322         # Find the most similar items to this one
323         scores = botMatches(prefs, item, n, similarity=sim_distance)
324         result[item] = scores
325     return result
326
327 def loadMovie(path='data'): #movies, with per user ratings
328   # Get movie titles
329     movies = getMovieTitles()
330   # Load data
331     prefs = {}
332     for line in open(path + '/u.data'):
333         (user, movieid, rating, ts) = line.split('\t')
334         prefs.setdefault(movieid, {})
335         prefs[movieid][user] = float(rating)
336     return prefs
337
338 def getTopGunCorrelation(path='data'): #get movies most alike and unlike Top Gun
339     #load list of movies, with per user ratings
340     mList = loadMovie()
341     topGun = 0
342      # Get movie titles
343     movies = getMovieTitles()
344     for m in movies:
345         if 'Top Gun' in movies[m]:
346             topGun = m
347     sim = calculateSimilarItems(mList, 1)
348     neg = calculateDissimilarItems(mList, 1)
349     print '\nMovie most like Top Gun: '
350     print movies[sim[m][0][1]]
351     print '\nMovie least like Top Gun: '
352     print movies[neg[m][0][1]]
353 #-----------------------------------------------------------
354 def loadUserRatings(path='data'): # users with per movie ratings
355   # Get movie titles
356     movies = getMovieTitles()
357   # Load data
358     prefs = {}
359     for line in open(path + '/u.data'):
360         (user, movieid, rating, ts) = line.split('\t')
361         prefs.setdefault(user, {})
362         prefs[user][movieid] = float(rating)
```

```
363     return prefs
364
365 def getTopRaters(n=5):  #get users that rated the most movies
366     users = loadUserRatings()
367     rsort = sorted(users.items(), key=lambda x: len(x[1]), reverse=True)
368     print '\nTop 5 users by most ratings:'
369     print string.ljust('User', 12), string.rjust('Number of Ratings', 3)
370     print string.ljust('------', 12), string.rjust('----------------', 3)
371     for x in range(0, n):
372         print string.ljust(rsort[x][0], 12), string.rjust(str(len(rsort[x][1])), 3)
373 #-----------------------------------------------------------------------------
374 def getUserCorrelation(n=5):  #get users that are most correlated with each other
375     '''
376     Finds the 5 users that agree the most by finding the users that are closest
377     '''
378     users = loadUserRatings()
379     alike = calculateSimilarItems(users, 1)
380     min = float('inf')
381     minid = 0
382     chain = {}
383     r = {}
384     for a in alike:
385         sum = 0
386         c = alike[a]
387         chain[a] = {}
388         links = []
389         links.append(a)
390         r[a] = [a, alike[a][0][1], sim_pearson(users, str(a), alike[a][0][1]), sim_distance(users, str(a),
                 alike[a][0][1])]
391         sum = sum + sim_distance(users, str(a), alike[a][0][1])
392         for i in range(0, n):
393             add = links[i]
394             links.append(alike[add][0][1])
395             next = alike[alike[add][0][1]][0][1]
396             sum = sum + sim_distance(users, alike[add][0][1], alike[alike[add][0][1]][0][1])
397             chain[a]['links'] = list(set(links))
398         chain[a]['distance'] = sum
399     for c in chain:
400         if len(chain[c]['links']) >= 5:
401             if chain[c]['distance'] < min:
402                 min = chain[c]['distance']
403                 minid = c
404     chain[minid]['r'] = {}
405     for i in range(0, len(chain[minid]['links'])):
406         chain[minid]['r'][i] = r[chain[minid]['links'][i]]
407     csort = sorted(chain[minid]['r'].items(), key=lambda x: x[1][2], reverse=True)
408     print '5 closest raters with r values and distance: '
409     for c in csort:
410         print 'User ' + c[1][0] + ' to ' + c[1][1] + ', r = ' + str(c[1][2]) + ', distance= ' + str(c
              [1][3])
411     #Finds the 5 users that disagree the most by finding the users that are farthest away
412     unlike = calculateDissimilarItems(users, 1)
413     max = float('-inf')
414     maxid = 0
415     uchain = {}
416     ur = {}
417     for u in unlike:
418         sum = 0
419         c = unlike[u]
420         uchain[u] = {}
421         links = []
422         links.append(u)
423         ur[u] = [a, unlike[u][0][1], sim_pearson(users, str(u), unlike[u][0][1]), sim_distance(users, str(u
                 ), unlike[u][0][1])]
424         sum = sum + sim_distance(users, str(u), unlike[u][0][1])
425         for i in range(0, n):
426             add = links[i]
427             links.append(unlike[add][0][1])
428             next = unlike[unlike[add][0][1]][0][1]
429             sum = sum + sim_distance(users, unlike[add][0][1], unlike[unlike[add][0][1]][0][1])
430             uchain[u]['links'] = list(set(links))
431         uchain[u]['distance'] = sum
432     for c in uchain:
```

```
433         if len(uchain[c]['links']) == 5:
434             if uchain[c]['distance'] > max:
435                 max = uchain[c]['distance']
436                 maxid = c
437     uchain[maxid]['ur'] = {}
438     for i in range(0, len(uchain[maxid]['links'])):
439         uchain[maxid]['ur'][i] = ur[uchain[maxid]['links'][i]]
440     csort = sorted(uchain[maxid]['ur'].items(), key=lambda x: x[1][2], reverse=True)
441     print '5 farthest raters, with r values and distance: '
442     for c in csort:
443         print 'User ' + c[1][0] + ' to ' + c[1][1] + ', r = ' + str(c[1][2]) + ', distance= ' + str(c
              [1][3])
444
445 #--------------------------------------------------------------
446 def getRatingsByAge(n=5):  #get top average, most rated, ratings by gender
447     rate = loadMovieRatings()
448     users = loadUserInfo()
449     women = {}
450     w40 = {}
451     men = {}
452     m40 = {}
453     for r in rate:
454         women[r] = {}
455         men[r] = {}
456         w40[r] = {}
457         m40[r] = {}
458     for u in users:
459         if users[u]['info']['gender'] == 'F':
460             if int(users[u]['info']['age']) < 40:
461                 for r in rate:
462                     if users[u].has_key(r):
463                         if women[r].has_key('ratings'):
464                             women[r]['ratings'].append(users[u][r])
465                         else:
466                             women[r]['ratings'] = []
467                             women[r]['ratings'].append(users[u][r])
468             else:
469                 for r in rate:
470                     if users[u].has_key(r):
471                         if w40[r].has_key('ratings'):
472                             w40[r]['ratings'].append(users[u][r])
473                         else:
474                             w40[r]['ratings'] = []
475                             w40[r]['ratings'].append(users[u][r])
476         else:
477             if int(users[u]['info']['age']) < 40:
478                 for r in rate:
479                     if users[u].has_key(r):
480                         if men[r].has_key('ratings'):
481                             men[r]['ratings'].append(users[u][r])
482                         else:
483                             men[r]['ratings'] = []
484                             men[r]['ratings'].append(users[u][r])
485             else:
486                 for r in rate:
487                     if users[u].has_key(r):
488                         if m40[r].has_key('ratings'):
489                             m40[r]['ratings'].append(users[u][r])
490                         else:
491                             m40[r]['ratings'] = []
492                             m40[r]['ratings'].append(users[u][r])
493     # Get the average ratings per movie per gender
494     for w in women:
495         if women[w].has_key('ratings'):
496             rate[w]['women'] = ratingAverage(women[w])
497         else:
498             rate[w]['women'] = float('-inf')
499     for m in men:
500         if men[m].has_key('ratings'):
501             rate[m]['men'] = ratingAverage(men[m])
502         else:
503             rate[m]['men'] = float('-inf')
504     for w in w40:
```

```
505         if w40[w].has_key('ratings'):
506             rate[w]['w40'] = ratingAverage(w40[w])
507         else:
508             rate[w]['w40'] = float('-inf')
509     for m in m40:
510         if m40[m].has_key('ratings'):
511             rate[m]['m40'] = ratingAverage(m40[m])
512         else:
513             rate[m]['m40'] = float('-inf')
514     # Print results per gender
515     print '\nTop ' + str(n) + ' movies Rated by Men under 40:'
516     printRatings(rate.items(), 'men', n)
517     print '\nTop ' + str(n) + ' movies Rated by Men over 40:'
518     printRatings(rate.items(), 'm40', n)
519     print '\nTop ' + str(n) + ' movies Rated by Women under 40:'
520     printRatings(rate.items(), 'women', n)
521     print '\nTop ' + str(n) + ' movies Rated by Women over 40:'
522     printRatings(rate.items(), 'w40', n)
```

Listing 8: Python Recommendations

REFERENCES

[1] Alok Singhal. sorting a nested dictionary with lists in python. http://stackoverflow.com/questions/14719654/sorting-a-nested-dictionary-with-lists-in-python. Accessed: 2014-11-8.

[2] David Beazley Brian K. Jones. *Python Cookbook*. O'Reilly Media, 3rd edition, 2013.

[3] Fancier Output Formatting. https://docs.python.org/release/1.5.1p1/tut/node50.html. Accessed: 2014-11-9.

[4] Toby Segaran. *Programming Collective Intelligence*. O'Reilly Media, 2007.