INTRODUCTION TO WEB SCIENCE- FALL 2014

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

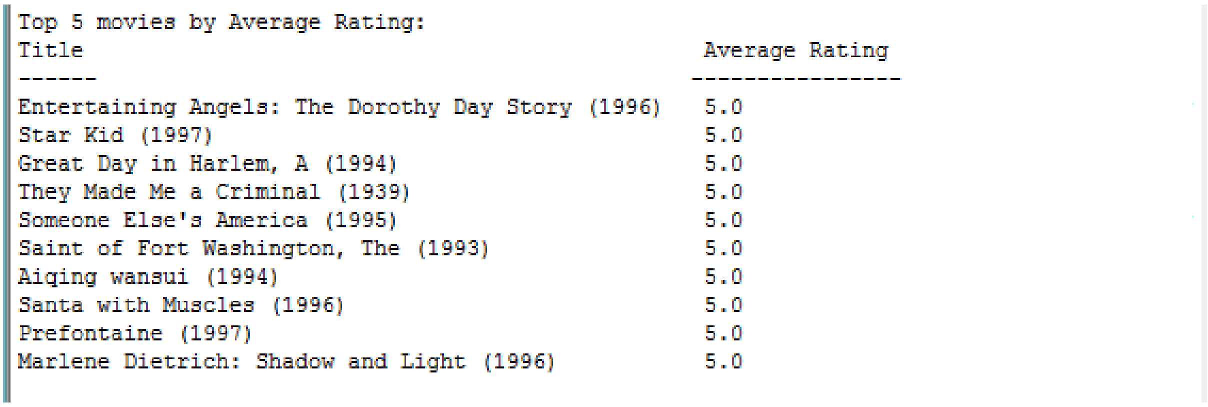
The first step was to load the data into a usable structure. The movie title to id mappings were used frequently, so getMovieTitles was written from the loadMovieLens function, to be reusable and returns a dictionary of the mappings. Module loadMovieRatings is a modified version of loadMovieLens. It 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.

```
def getMovieTitles(path='data'):
  # Get movie titles
    movies = {}
    for line in open(path + '/u.item'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
    return movies

def loadMovieRatings(path='data'):  #load movies with title and ratings list
    # Get movie titles
    movies = getMovieTitles()
    # Get all ratings for each movie
    rate = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        rate.setdefault(movieid, {})
        rate[movieid]['title'] = movies[movieid]
        if rate[movieid].has_key('ratings'):
            rate[movieid]['ratings'].append(float(rating))
        else:
            rate[movieid]['ratings'] = []
            rate[movieid]['ratings'].append(float(rating))
    return rate

def ratingAverage(ratings):  #get average of a list of movie ratings
    sum = 0.0
    for i in range(0, len(ratings['ratings'])):
        sum = sum + ratings['ratings'][i]
    avg = sum / len(ratings['ratings'])
    return avg

def getTop(sort, type, n=5):
    max = sort[0][1][type]
    top = []
    top.append(sort[0])
    for s in range(1, len(sort)):
        if sort[s][1][type] == max:
            top.append(sort[s])
        else:
            break
    while len(top) < n:
        max = sort[len(top)][1][type]
        for s in range(len(top), len(sort)):
            if sort[s][1][type] == max:
                top.append(sort[s])
            else:
                break
    return top

def printRatings(items, type, n):  #print list of ratings
    sort = sorted(items, key=lambda x: x[1][type], reverse=True)
    col_width = 0
    top = getTop(sort, type, n)
    # Get longest title for printed column width
    for x in range(0, len(top)):
        if len(top[x][1]['title']) > col_width:
            col_width = len(top[x][1]['title']) + 2
    # print the result
    print string.ljust('Title', col_width), string.rjust('Average Rating', 3)
    print string.ljust('\n------', col_width), string.rjust('----------------', 3)
    for x in range(0, len(top)):
        print string.ljust(top[x][1]['title'], col_width), string.rjust(str(top[x][1][type]), 3)

def getRatings(n=5):  #get top average, most rated
    rate = loadMovieRatings()
    # Get the average ratings per movie
    for r in rate:
        rate[r]['average'] = ratingAverage(rate[r])
    print 'Top ' + str(n) + ' movies by Average Rating:'
    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

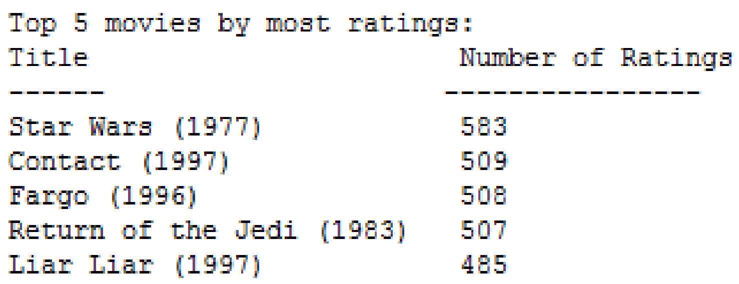
The same function, getRatings, was used as inthe first question. The data was sorted differently, on the length of the ratings list, for each movie. Since the sorting had to be completed by the rating list length, instead of an average value , the print statements had to be written separately. Listing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

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

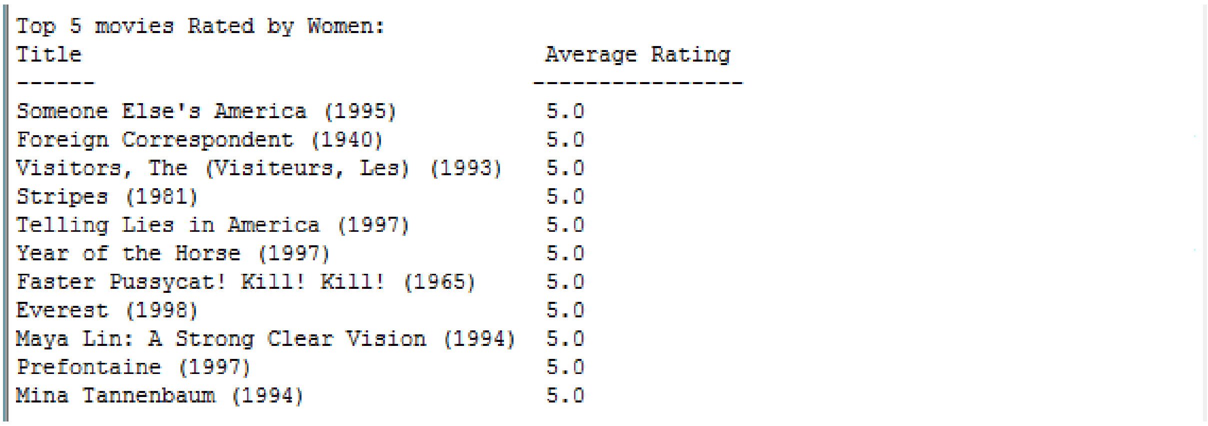
A separate module was written for the gender classifications, because additional user information was needed. Module loadUserInfo placed the user id to movie rating mappings with a nested dictionary with age and gender. [1] The loadMovieRatings module was used again, for consistency and reuse of average 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.

```
def loadUserInfo(path='data'):  #load users with info and movie ratings
  # Load data
    prefs = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movieid] = float(rating)
    for line in open(path + '/u.user'):
        (user, age, gender, job, zip) = line.split('|')
        prefs[user]['info'] = {}
        prefs[user]['info']['age'] = int(age)
        prefs[user]['info']['gender'] = gender
    return prefs

def getRatingsByGender(n=5):  #get top average, most rated, ratings by gender
    rate = loadMovieRatings()
        # Get ratings per gender
    users = loadUserInfo()
    women = {}
    men = {}
    for r in rate:
        women[r] = {}
        men[r] = {}
    for u in users:
        if users[u]['info']['gender'] == 'F':
            for r in rate:
                if users[u].has_key(r):
                    if women[r].has_key('ratings'):
                        women[r]['ratings'].append(users[u][r])
                    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

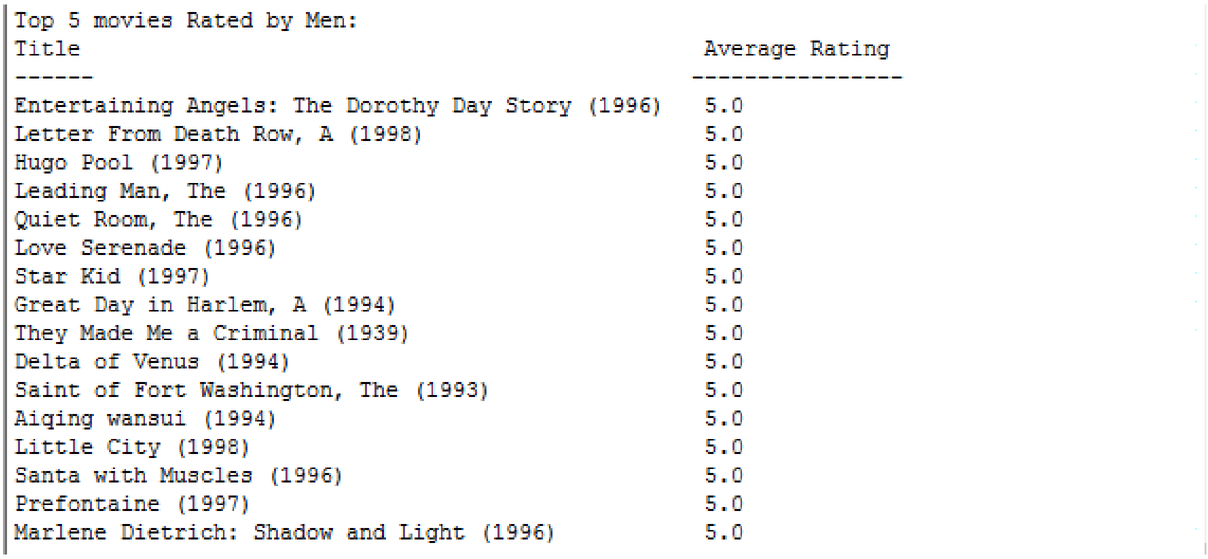
What 5 movies were rated the highest on average by men? Show the movies and their ratings sorted by ratings.

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.

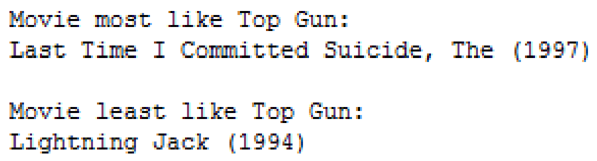
```
def topMatches(prefs, original, n=5, similarity=sim_pearson):
    '''
    Returns the best matches for an item from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, original, other), other) for other in prefs
      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

```
     # Get movie titles
    movies = getMovieTitles()
    for m in movies:
        if 'Top Gun' in movies[m]:
            topGun = m
    sim = calculateSimilarItems(mList, 1)
    neg = calculateDissimilarItems(mList, 1)
    print '\nMovie most like Top Gun: '
    print movies[sim[m][0][1]]
    print '\nMovie least like Top Gun: '
    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

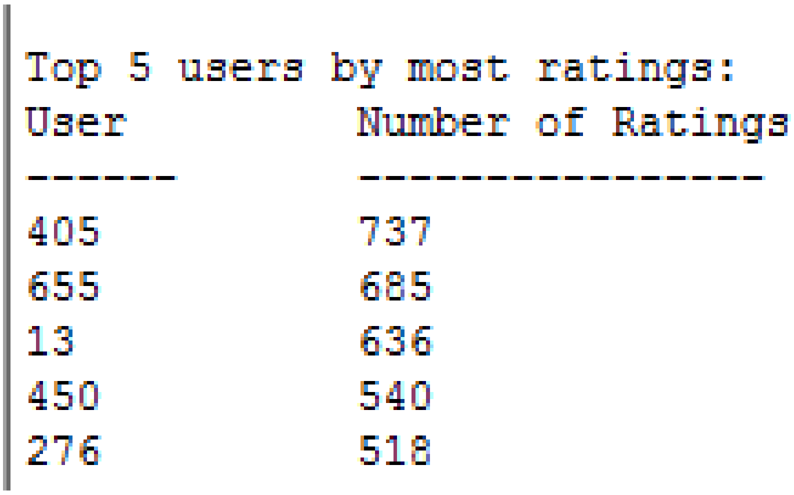
A new module, loadUserRatings, was written to compile a dictionary of users with a list of their movie ratings. Using this, the top raters were found by using the length of the list of movie ratings. Listing 5 shows the loadUserRatings and getTopRaters module.

```
def loadUserRatings(path='data'): # users with per movie 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(user, {})
        prefs[user][movieid] = float(rating)
    return prefs

def getTopRaters(n=5):  #get users that rated the most movies
    users = loadUserRatings()
    rsort = sorted(users.items(), key=lambda x: len(x[1]), reverse=True)
    print '\nTop 5 users by most ratings:'
    print string.ljust('User', 12), string.rjust('Number of Ratings', 3)
    print string.ljust('------', 12), string.rjust('-----------------', 3)
    for x in range(0, n):
        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.

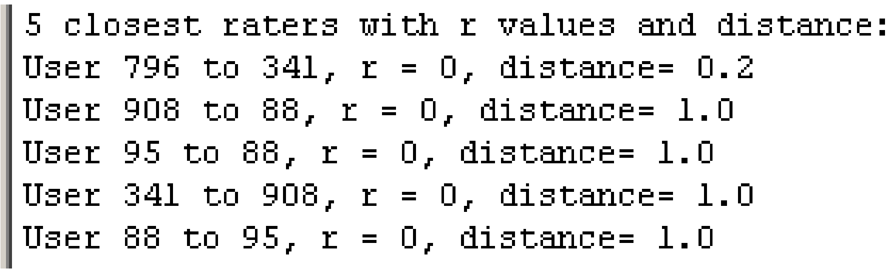
1.7.1 Solution

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.

```python
def getUserCorrelation(n=5):  #get users that are most correlated with each other
    '''
    Finds the 5 users that agree the most by finding the users that are closest
    '''
    users = loadUserRatings()
    alike = calculateSimilarItems(users, 1)
    min = float('inf')
    minid = 0
    chain = {}
    r = {}
    for a in alike:
        sum = 0
        c = alike[a]
        chain[a] = {}
        links = []
        links.append(a)
        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])]
        sum = sum + sim_distance(users, str(a), alike[a][0][1])
        for i in range(0, n):
            add = links[i]
            links.append(alike[add][0][1])
            next = alike[alike[add][0][1]][0][1]
            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

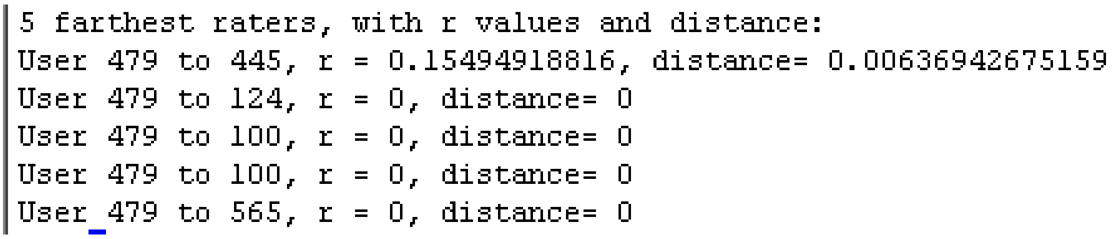
The code to find the 5 raters who disagreed was the same as the code for agreement, except using calcuateDissimilarItems. Listing 6 is the additional code in getUserCorrelation.

```
    #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

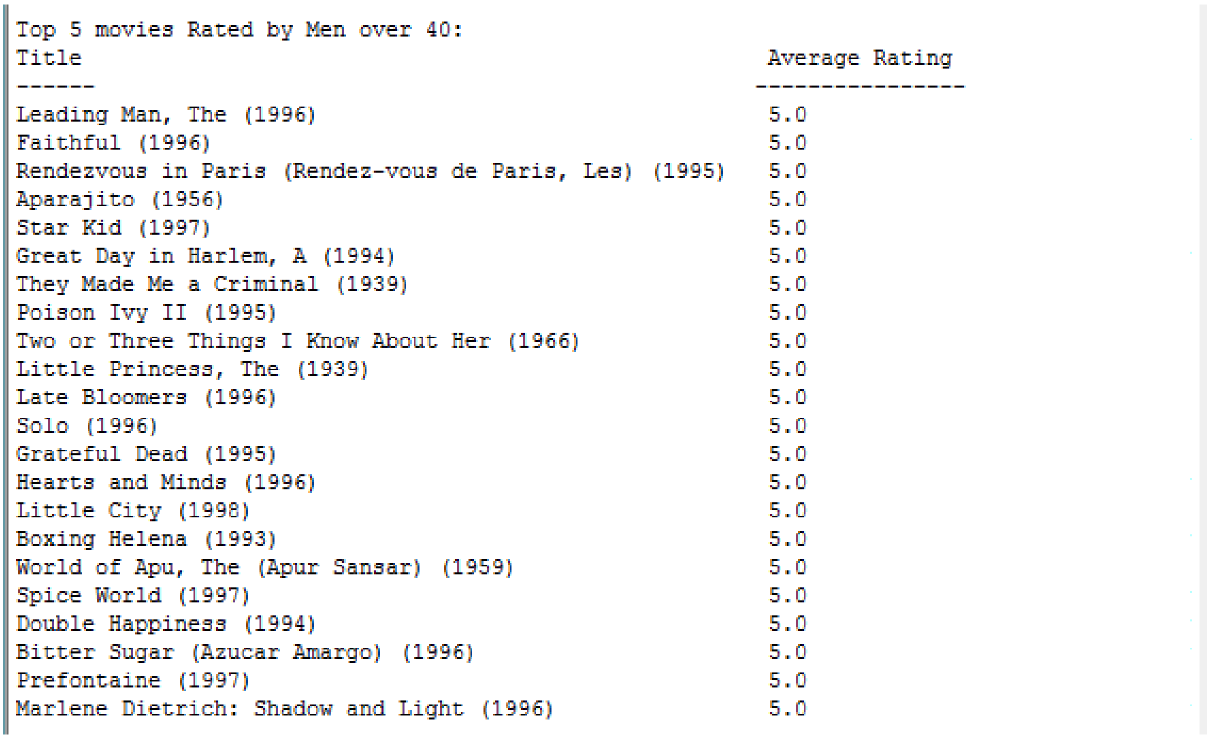
This solution was similar to the gender based solution. The modules used to import the data were loadMovieRatings from listing 1 and loadUserInfo from listing 3. Dictionaries were created for both gender based age groups, so the averages could be calculated independently. Then for each user, gender and 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] = {}
        m40[r] = {}
    for u in users:
        if users[u]['info']['gender'] == 'F':
            if int(users[u]['info']['age']) < 40:
                for r in rate:
                    if users[u].has_key(r):
                        if women[r].has_key('ratings'):
                            women[r]['ratings'].append(users[u][r])
                        else:
                            women[r]['ratings'] = []
                            women[r]['ratings'].append(users[u][r])
            else:
                for r in rate:
                    if users[u].has_key(r):
                        if w40[r].has_key('ratings'):
                            w40[r]['ratings'].append(users[u][r])
                        else:
                            w40[r]['ratings'] = []
                            w40[r]['ratings'].append(users[u][r])
        else:
            if int(users[u]['info']['age']) < 40:
                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])
            else:
                for r in rate:
                    if users[u].has_key(r):
                        if m40[r].has_key('ratings'):
                            m40[r]['ratings'].append(users[u][r])
                        else:
                            m40[r]['ratings'] = []
                            m40[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])
        else:
            rate[w]['women'] = float('-inf')
    for m in men:
        if men[m].has_key('ratings'):
            rate[m]['men'] = ratingAverage(men[m])
        else:
            rate[m]['men'] = float('-inf')
    for w in w40:
        if w40[w].has_key('ratings'):
            rate[w]['w40'] = ratingAverage(w40[w])
        else:
            rate[w]['w40'] = float('-inf')
    for m in m40:
        if m40[m].has_key('ratings'):
            rate[m]['m40'] = ratingAverage(m40[m])
        else:
            rate[m]['m40'] = float('-inf')
    # Print results per gender
    print '\nTop ' + str(n) + ' movies Rated by Men under 40:'
    printRatings(rate.items(), 'men', n)
    print '\nTop ' + str(n) + ' movies Rated by Men over 40:'
    printRatings(rate.items(), 'm40', n)
    print '\nTop ' + str(n) + ' movies Rated by Women under 40:'
    printRatings(rate.items(), 'women', n)
    print '\nTop ' + str(n) + ' movies Rated by Women over 40:'
    printRatings(rate.items(), 'w40', n)
```

Listing 7: Ratings by Age

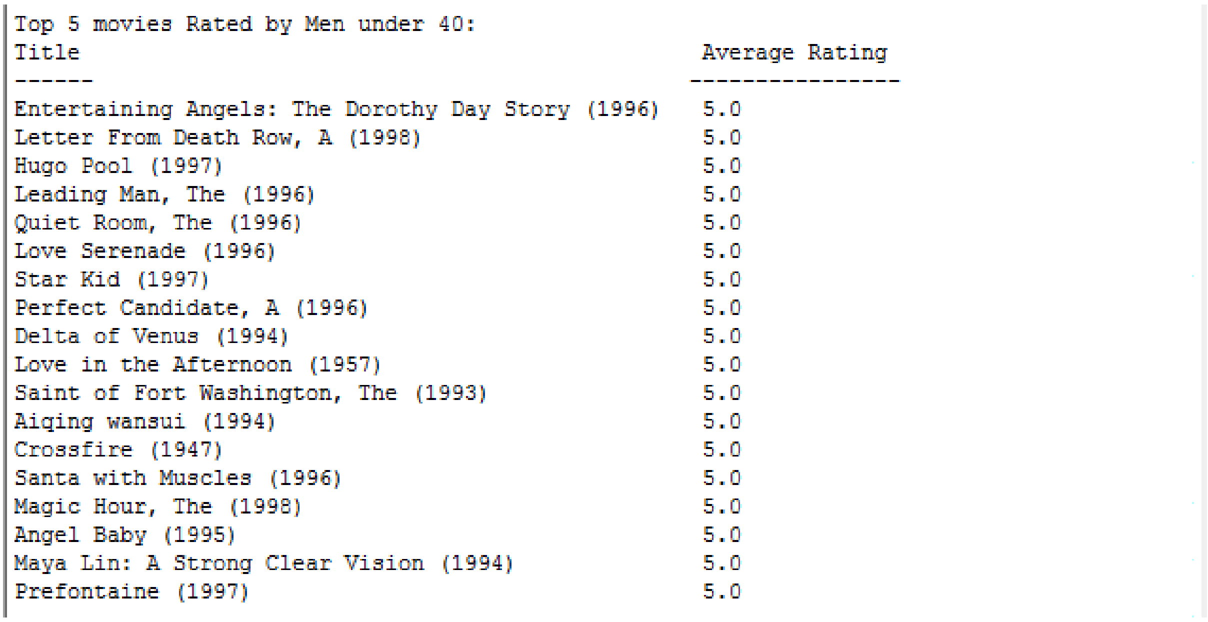
1.9.2 Result

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

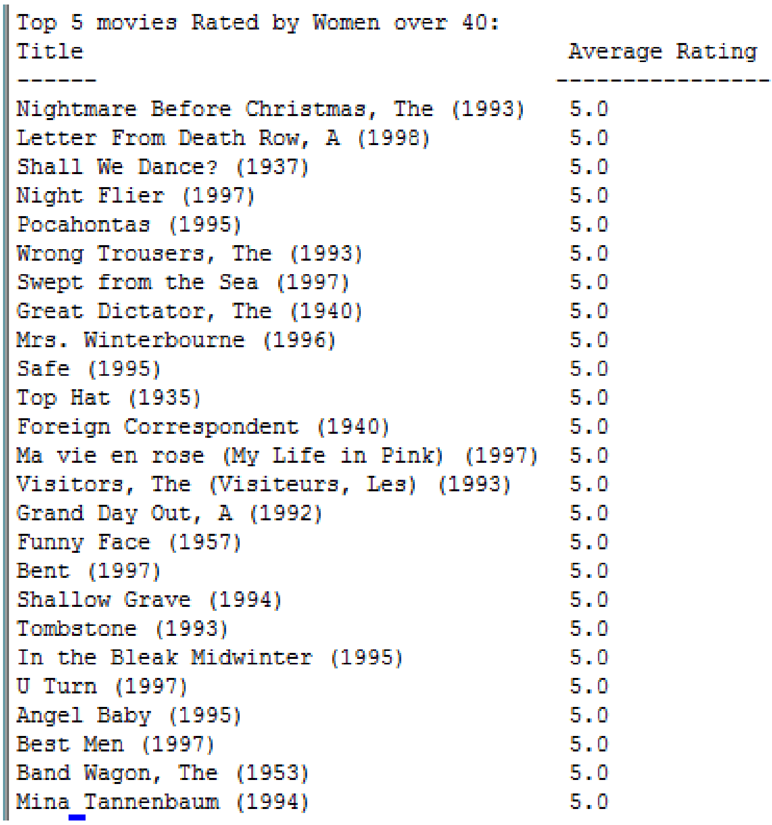
What movie was rated highest on average by women over 40? By women under 40?

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.

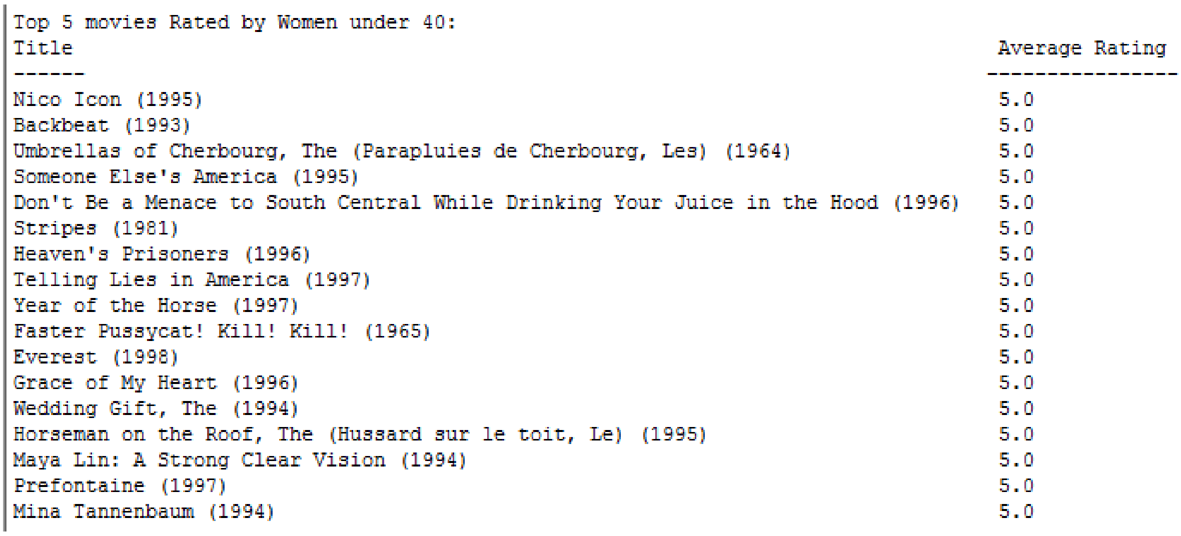
1.10.2 Result

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

```python
#!/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:
            # Similarity * Score
            totals.setdefault(item, 0)
            # The final score is calculated by multiplying each item by the
            #   similarity and adding these products together
            totals[item] += prefs[other][item] * sim
            # Sum of similarities
            simSums.setdefault(item, 0)
            simSums[item] += sim
    # Create the normalized list
    rankings = [(total / simSums[item], item) for (item, total) in
                totals.items()]
    # Return the sorted list
    rankings.sort()
    rankings.reverse()
    return rankings

def transformPrefs(prefs):
    ’’’
    Transform the recommendations into a mapping where persons are described
    with interest scores for a given title e.g. {title: person} instead of
    {person: title}.
    ’’’
    result = {}
    for person in prefs:
        for item in prefs[person]:
            result.setdefault(item, {})
            # Flip item and person
            result[item][person] = prefs[person][item]
    return result

def getRecommendedItems(prefs, itemMatch, user):
    userRatings = prefs[user]
    scores = {}
    totalSim = {}
    # Loop over items rated by this user
    for (item, rating) in userRatings.items():
        # Loop over items similar to this one
        for (similarity, item2) in itemMatch[item]:
            # Ignore if this user has already rated this item
            if item2 in userRatings:
                continue
            # Weighted sum of rating times similarity
            scores.setdefault(item2, 0)
            scores[item2] += similarity * rating
            # Sum of all the similarities
            totalSim.setdefault(item2, 0)
            totalSim[item2] += similarity
    # Divide each total score by total weighting to get an average
    rankings = [(score / totalSim[item], item) for (item, score) in
                scores.items()]
    # Return the rankings from highest to lowest
    rankings.sort()
    rankings.reverse()
    return rankings
#---------------------------------------------------------------------------
def getMovieTitles(path=’data’):
  # Get movie titles
    movies = {}
    for line in open(path + ’/u.item’):
        (id, title) = line.split(’|’)[0:2]
        movies[id] = title
    return movies

def loadMovieRatings(path=’data’):  #load movies with title and ratings list
    # Get movie titles
    movies = getMovieTitles()
    # Get all ratings for each movie
    rate = {}
    for line in open(path + ’/u.data’):
        (user, movieid, rating, ts) = line.split(’\t’)
        rate.setdefault(movieid, {})
        rate[movieid][’title’] = movies[movieid]
        if rate[movieid].has_key(’ratings’):
            rate[movieid]['ratings'].append(float(rating))
        else:
            rate[movieid]['ratings'] = []
            rate[movieid]['ratings'].append(float(rating))
    return rate

def ratingAverage(ratings):  #get average of a list of movie ratings
    sum = 0.0
    for i in range(0, len(ratings['ratings'])):
        sum = sum + ratings['ratings'][i]
    avg = sum / len(ratings['ratings'])
    return avg

def getTop(sort, type, n=5):
    max = sort[0][1][type]
    top = []
    top.append(sort[0])
    for s in range(1, len(sort)):
        if sort[s][1][type] == max:
            top.append(sort[s])
        else:
            break
    while len(top) < n:
        max = sort[len(top)][1][type]
        for s in range(len(top), len(sort)):
            if sort[s][1][type] == max:
                top.append(sort[s])
            else:
                break
    return top

def printRatings(items, type, n):  #print list of ratings
    sort = sorted(items, key=lambda x: x[1][type], reverse=True)
    col_width = 0
    top = getTop(sort, type, n)
    # Get longest title for printed column width
    for x in range(0, len(top)):
        if len(top[x][1]['title']) > col_width:
            col_width = len(top[x][1]['title']) + 2
    # print the result
    print string.ljust('Title', col_width), string.rjust('Average Rating', 3)
    print string.ljust('\n------', col_width), string.rjust('----------------', 3)
    for x in range(0, len(top)):
        print string.ljust(top[x][1]['title'], col_width), string.rjust(str(top[x][1][type]), 3)

def getRatings(n=5):  #get top average, most rated
    rate = loadMovieRatings()
    # Get the average ratings per movie
    for r in rate:
        rate[r]['average'] = ratingAverage(rate[r])
    print 'Top ' + str(n) + ' movies by Average Rating:'
    printRatings(rate.items(), 'average', n)
    # Get the most rated movies
    rsort = sorted(rate.items(), key=lambda x: len(x[1]['ratings']), reverse=True)
    print '\nTop ' + str(n) + ' movies by most ratings:'
    col_width = 0
    for x in range(0, n):
        if len(rsort[x][1]['title']) > col_width:
            col_width = len(rsort[x][1]['title']) + 2
    print string.ljust('Title', col_width), string.rjust('Number of Ratings', 3)
    print string.ljust('------', col_width), string.rjust('----------------', 3)
    for x in range(0, n):
        print string.ljust(rsort[x][1]['title'], col_width), string.rjust(str(len(rsort[x][1]['ratings'])),
            3)
#------------------------------------------------------------------------
def loadUserInfo(path='data'):  #load users with info and movie ratings
  # Load data
    prefs = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movieid] = float(rating)
    for line in open(path + '/u.user'):
        (user, age, gender, job, zip) = line.split('|')
        prefs[user]['info'] = {}
        prefs[user]['info']['age'] = int(age)
        prefs[user]['info']['gender'] = gender
    return prefs

def getRatingsByGender(n=5):  #get top average, most rated, ratings by gender
    rate = loadMovieRatings()
        # Get ratings per gender
    users = loadUserInfo()
    women = {}
    men = {}
    for r in rate:
        women[r] = {}
        men[r] = {}
    for u in users:
        if users[u]['info']['gender'] == 'F':
            for r in rate:
                if users[u].has_key(r):
                    if women[r].has_key('ratings'):
                        women[r]['ratings'].append(users[u][r])
                    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)
#------------------------------------------------------------
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
     # Get movie titles
    movies = getMovieTitles()
    for m in movies:
        if 'Top Gun' in movies[m]:
            topGun = m
    sim = calculateSimilarItems(mList, 1)
    neg = calculateDissimilarItems(mList, 1)
    print '\nMovie most like Top Gun: '
    print movies[sim[m][0][1]]
    print '\nMovie least like Top Gun: '
    print movies[neg[m][0][1]]
#-----------------------------------------------------------------------
def loadUserRatings(path='data'): # users with per movie 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(user, {})
        prefs[user][movieid] = float(rating)
```

```python
    return prefs

def getTopRaters(n=5):  #get users that rated the most movies
    users = loadUserRatings()
    rsort = sorted(users.items(), key=lambda x: len(x[1]), reverse=True)
    print '\nTop 5 users by most ratings:'
    print string.ljust('User', 12), string.rjust('Number of Ratings', 3)
    print string.ljust('------', 12), string.rjust('-----------------', 3)
    for x in range(0, n):
        print string.ljust(rsort[x][0], 12), string.rjust(str(len(rsort[x][1])), 3)
#-----------------------------------------------------------------------------
def getUserCorrelation(n=5):  #get users that are most correlated with each other
    '''
    Finds the 5 users that agree the most by finding the users that are closest
    '''
    users = loadUserRatings()
    alike = calculateSimilarItems(users, 1)
    min = float('inf')
    minid = 0
    chain = {}
    r = {}
    for a in alike:
        sum = 0
        c = alike[a]
        chain[a] = {}
        links = []
        links.append(a)
        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])]
        sum = sum + sim_distance(users, str(a), alike[a][0][1])
        for i in range(0, n):
            add = links[i]
            links.append(alike[add][0][1])
            next = alike[alike[add][0][1]][0][1]
            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])
    #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])

#-----------------------------------------------------------------------
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] = {}
        m40[r] = {}
    for u in users:
        if users[u]['info']['gender'] == 'F':
            if int(users[u]['info']['age']) < 40:
                for r in rate:
                    if users[u].has_key(r):
                        if women[r].has_key('ratings'):
                            women[r]['ratings'].append(users[u][r])
                        else:
                            women[r]['ratings'] = []
                            women[r]['ratings'].append(users[u][r])
            else:
                for r in rate:
                    if users[u].has_key(r):
                        if w40[r].has_key('ratings'):
                            w40[r]['ratings'].append(users[u][r])
                        else:
                            w40[r]['ratings'] = []
                            w40[r]['ratings'].append(users[u][r])
        else:
            if int(users[u]['info']['age']) < 40:
                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])
            else:
                for r in rate:
                    if users[u].has_key(r):
                        if m40[r].has_key('ratings'):
                            m40[r]['ratings'].append(users[u][r])
                        else:
                            m40[r]['ratings'] = []
                            m40[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])
        else:
            rate[w]['women'] = float('-inf')
    for m in men:
        if men[m].has_key('ratings'):
            rate[m]['men'] = ratingAverage(men[m])
        else:
            rate[m]['men'] = float('-inf')
    for w in w40:
```

```python
        if w40[w].has_key('ratings'):
            rate[w]['w40'] = ratingAverage(w40[w])
        else:
            rate[w]['w40'] = float('-inf')
    for m in m40:
        if m40[m].has_key('ratings'):
            rate[m]['m40'] = ratingAverage(m40[m])
        else:
            rate[m]['m40'] = float('-inf')
    # Print results per gender
    print '\nTop ' + str(n) + ' movies Rated by Men under 40:'
    printRatings(rate.items(), 'men', n)
    print '\nTop ' + str(n) + ' movies Rated by Men over 40:'
    printRatings(rate.items(), 'm40', n)
    print '\nTop ' + str(n) + ' movies Rated by Women under 40:'
    printRatings(rate.items(), 'women', n)
    print '\nTop ' + str(n) + ' movies Rated by Women over 40:'
    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.