Bo Zhang
01063214

1 3 Closest Users

Find 3 users who are closest to you in terms of age, gender, and occupation. For each of those 3 users:
- what are their top 3 favorite films?
- bottom 3 least favorite films?

Algorithm:

1. Open the user data.
2. For each user whose gender is male and occupation is student, calculate the difference of age with me.
3. Sort all the users above with the difference of age.
4. Use the script from "Programming Collective Intelligence" to open the rating data.
5. For the top 3 users, sort their ratings.
6. Output the top 3 and bottom 3 films of them.

Source code:

Listing 1: The content of Q1.py

```
def loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def loadMovieLens(movies,path='ml-100k'):
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

def loadUsers(path='ml-100k'):
    users={}
```

```
    for line in open(path+'/u.user', 'r', encoding='ISO-8859-1'):
        (user,age,gender,occupation,zipcode)=line.split('|')
        users.setdefault(user,{})
        users[user]['age']=age
        users[user]['gender']=gender
        users[user]['occupation']=occupation
        users[user]['zipcode']=zipcode
    return users

users = loadUsers()
scores=[(abs(int(users[user]['age'])-32),user)
        for user in users if users[user]['gender']=='M' and users[user]['occupation']=='student']
scores.sort()
movies = loadMovies()
prefs = loadMovieLens(movies)

for i in range(3):
    films=[]
    for movie in prefs[scores[i][1]]:
        row = (prefs[scores[i][1]][movie],movie)
        films.append(row)
    films.sort()
    print ('\nUser'+scores[i][1]+'\'s bottom 3 least favorite films:')
    for j in range(3):
        print (films[j][1]+'\t{}'.format(films[j][0]))
    films.reverse()
    print ('\nUser'+scores[i][1]+'\'s top 3 favorite films:')
    for j in range(3):
        print (films[j][1]+'\t{}'.format(films[j][0]))
```

Results:

Table 1: Top and Bottom Films of 3 Closest Users

User	Rank	Top 3	Rating	Bottom 3	Rating
	1	Wizard of Oz, The (1939)	5.0	Hunt for Red October, The (1990)	2.0
350	2	Wild Bunch, The (1969)	5.0	M*A*S*H (1970)	2.0
	3	Vertigo (1958)	5.0	Alien (1979)	3.0
	1	Usual Suspects, The (1995)	5.0	Bed of Roses (1996)	1.0
560	2	Star Wars (1977)	5.0	Event Horizon (1997)	1.0
	3	L.A. Confidential (1997)	5.0	Kids in the Hall: Brain Candy (1996)	1.0
	1	Young Frankenstein (1974)	5.0	Batman (1989)	1.0
890	2	Wizard of Oz, The (1939)	5.0	Ref, The (1994)	1.0
	3	Willy Wonka and the Chocolate Factory (1971)	5.0	Star Trek: The Motion Picture (1979)	1.0

I think user560 is most like me.

2 Most and Least Correlated Users

Which 5 users are most correlated to the substitute you? Which 5 users are least correlated?

Algorithm:

1. Use the script from "Programming Collective Intelligence" to open the rating data.
2. Edit the "topMatches' function to make it able to return either Top or Bottom matches.
3. Use the "topMatches" function to get the both 5 most correlated and 5 least correlated users.
4. Output the user list.

Source code:

Listing 2: The content of Q2.py

```
from math import sqrt

# Returns the Pearson correlation coefficient for p1 and p2
def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1

    # Find the number of elements
    n=len(si)

    # if they are no ratings in common, return 0
    if n==0: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up 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 up the products
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

    # Calculate 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

# Returns the best matches for person from the prefs dictionary.
# Number of results and similarity function are optional params.
def topMatches(prefs,person,reverse=False,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other),other)
            for other in prefs if other!=person]

    # Sort the list so the highest scores appear at the top
    scores.sort()
    if reverse==False:
        scores.reverse()
    return scores[0:n]

def loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def loadMovieLens(movies,path='ml-100k'):
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

movies = loadMovies()
prefs = loadMovieLens(movies)

output=topMatches(prefs,'560')
print ('5 Most Correlated Users:')
for i in range(5):
```

```
    print (output[i][1]+'\t{}'.format(output[i][0]))
output=topMatches(prefs,'560',True)
print ('\n5 Least Correlated Users:')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
```

Results:

Table 2: 5 Most Correlated and 5 Least Correlated Users

Rank	Most Correlated User	Correlation	Least Correlated User	Correlation
1	685	1	810	-1
2	925	1	208	-1
3	914	1	155	-1
4	856	1	631	-1
5	732	1	777	-0.976

3 Top 5 and Bottom 5 Recommendations

Compute ratings for all the films that the substitute you have not seen. Provide a list of the top 5 recommendations for films that the substitute you should see. Provide a list of the bottom 5 recommendations.

Algorithm:

1. Use the script from "Programming Collective Intelligence" to open the rating data.
2. Edit the "getRecommendations" function to make it able to return either Top n or Bottom n recommendations.
3. Use the "getRecommendations" function to get both Top 5 and Bottom 5 recommendation films.
4. Output the film list.

Source code:

Listing 3: The content of Q3.py

```
from math import sqrt

# Returns the Pearson correlation coefficient for p1 and p2
def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1

    # Find the number of elements
    n=len(si)
```

```
    # if they are no ratings in common, return 0
    if n==0: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up 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 up the products
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

    # Calculate 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

# Gets recommendations for a person by using a weighted average
# of every other user's rankings
def getRecommendations(prefs,person,reverse=False,n=5,similarity=sim_pearson):
    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)
                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()
    if reverse==False:
        rankings.reverse()
    return rankings[0:n]

def loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def loadMovieLens(movies,path='ml-100k'):
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

movies = loadMovies()
prefs = loadMovieLens(movies)

output=getRecommendations(prefs,'560')
print ('Top 5 Recommendations:')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
output=getRecommendations(prefs,'560',True)
print ('\nBottom 5 Recommendations:')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
```

Results:

Table 3: Top 5 and Bottom 5 Recommendation Films

Rank	Top Films	Rating	Bottom Films	Rating
1	Saint of Fort Washington, The (1993)	5.0	3 Ninjas: High Noon At Mega Mountain (1998)	1.0
2	They Made Me a Criminal (1939)	5.0	Amityville 1992: It's About Time (1992)	1.0
3	Star Kid (1997)	5.0	Amityville: A New Generation (1993)	1.0
4	Someone Else's America (1995)	5.0	Amityville: Dollhouse (1996)	1.0
5	Santa with Muscles (1996)	5.0	August (1996)	1.0

4 Most and Least Correlated Films

Choose your favorite and least favorite film from the data. For each film, generate a list of the top 5 most correlated and bottom 5 least correlated films. Based on your knowledge of the resulting films, do you agree with the results? In other words, do you personally like / dislike the resulting films?

Algorithm:

1. Use the script from "Programming Collective Intelligence" to open the rating data and transform it.
2. Edit the "topMatches' function to make it able to return either Top or Bottom matches.
3. Use the "topMatches" function to get the both 5 most correlated and 5 least correlated films.
4. Output the film list.

Source code:

Listing 4: The content of Q4.py

```
from math import sqrt

# Returns the Pearson correlation coefficient for p1 and p2
def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1

    # Find the number of elements
    n=len(si)

    # if they are no ratings in common, return 0
    if n==0: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up 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 up the products
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

    # Calculate 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

# Returns the best matches for person from the prefs dictionary.
# Number of results and similarity function are optional params.
def topMatches(prefs,person,reverse=False,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other),other)
            for other in prefs if other!=person]

    # Sort the list so the highest scores appear at the top
    scores.sort()
    if reverse==False:
        scores.reverse()
    return scores[0:n]

def transformPrefs(prefs):
    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 loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
```

```
        movies[id]['URL']=URL
    return movies

def loadMovieLens(movies,path='ml-100k'):
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

movies = loadMovies()
prefs = loadMovieLens(movies)
movies = transformPrefs(prefs)

output=topMatches(movies,'Forrest Gump (1994)')
print ('5 Most Correlated Films of Forrest Gump (1994):')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
output=topMatches(movies,'Forrest Gump (1994)',True)
print ('\n5 Least Correlated Films of Forrest Gump (1994):')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))

output=topMatches(movies,'Chairman of the Board (1998)')
print ('\n5 Most Correlated Films of Chairman of the Board (1998):')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
output=topMatches(movies,'Chairman of the Board (1998)',True)
print ('\n5 Least Correlated Films of Chairman of the Board (1998):')
for i in range(5):
    print (output[i][1]+'\t{}'.format(output[i][0]))
```

Results:

Table 4: Top 5 Most Correlated and Bottom 5 Least Correlated Films of "Forrest Gump"

Rank	Most Correlated Films	Correlation	Least Correlated Films	Correlation
1	Dream With the Fishes (1997)	1.0	1-900 (1994)	-1.0
2	Zeus and Roxanne (1997)	1.0	American Dream (1990)	-1.0
3	The Innocent (1994)	1.0	Broken English (1996)	-1.0
4	Sliding Doors (1998)	1.0	Caro Diario (Dear Diary) (1994)	-1.0
5	Safe Passage (1994)	1.0	Collectionneuse, La (1967)	-1.0

Table 5: Top 5 Most Correlated and Bottom 5 Least Correlated Films of "Chairman of the Board"

Rank	Most Correlated Films	Correlation	Least Correlated Films	Correlation
1	Wings of the Dove, The (1997)	1.0	Anna Karenina (1997)	-1.0
2	Washington Square (1997)	1.0	Hercules (1997)	-1.0
3	Twisted (1996)	1.0	Men in Black (1997)	-1.0
4	Rock, The (1996)	1.0	Mrs. Brown (Her Majesty, Mrs. Brown) (1997)	-1.0
5	Red Corner (1997)	1.0	My Best Friend's Wedding (1997)	-1.0

As a foreigner, I have never seen most of the movies in the database. So it's too hard for me to tell if the result is good based on these old movies. For "Forrest Gump"(It's really 1 of my favorite movies), all the top 5 and bottom 5 correlated films are those I have never seen before. So I can't say anything about the prediction. For "Chairman of the Board"(I just choose a movie with a really low rating as my least favorite movie and I guess I will not like it too), I have just seen 2 of them in the top 5 and bottom 5 correlated films. But either "Rock, The (1996)" in the top 5, or "Men in Black (1997)" in the bottom 5, is a good movie to me. So I don't think it's a good prediction.

5 MovieLense data v.s. IMDB data

Rank the 1,682 movies according to the 1997/1998 MovieLense data. Now rank the same 1,682 movies according to todays IMDB data.

Draw a graph, where each dot is a film. The x-axis is the MovieLense ranking and the y-axis is today's IMDB ranking.

What is Pearon's r for the two lists? Assuming the two user bases are interchangable, what does this say about the attitudes about the films after nearly 20 years?

Algorithm:

1. Use the script from "Programming Collective Intelligence" to open the movie data.
2. For each movie, open the URL to get the rating and voters and save them into "IMDBrating.csv".
3. Open "IMDBrating.csv" and use the script from "Programming Collective Intelligence" to open the MovieLense rating data and transform it.
4. For each movie in "IMDBrating.csv", compute the voters and average rating of users in MovieLense data.
5. Sort the IMDB rating data and MovieLense rating data by rating and voters.
6. Replace the IMDB rating data and MovieLense rating data with movie ID and rank.
7. For each movie in "IMDBrating.csv", save its ID, MovieLense rank and IMDB rank into "rankContrast.csv".
8. Open "rankContrast.csv" in R, plot the scatter and run the Pearson's correlation test.

Source code:

Listing 5: The content of IMBDdownload.py

```
import urllib.request
from bs4 import BeautifulSoup

def loadMovies(path='ml-100k'):
    movies={}
```

```
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

movies=loadMovies()
for i in range(len(movies)):
    try:
        with urllib.request.urlopen(movies['%d' %(i+1)]['URL']) as res:
            html = res.read()
        soup = BeautifulSoup(html, "html.parser")
        line = soup.strong['title'].split('(')
        f = open('IMDBrating.csv','a',encoding='ISO-8859-1')
        f.write('{}'.format(i+1)+'\t'+line[0]+'\t'+line[3]+'\n')
        f.close()
    except:
        f = open('missingMovies.csv','a',encoding='ISO-8859-1')
        f.write('{}'.format(i+1)+'\n')
        f.close()
```

Listing 6: The content of Q5.py

```
def transformPrefs(prefs):
    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 loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
```

```
            (id,title)=line[0:2]
            URL=line[4]
            movies.setdefault(id,{})
            movies[id]['title']=title
            movies[id]['URL']=URL
        return movies

def loadMovieLens(movies,path='ml-100k'):
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

movies = loadMovies()
prefs = loadMovieLens(movies)
films = transformPrefs(prefs)

f = open('IMDBrating.csv', 'r', encoding='UTF-8')
lines = f.readlines()
f.close()
MovieLense=[]
IMDB=[]

for line in lines:
    row = line.strip().split('\t')
    rating = 0
    for item in films[movies[row[0]]['title']]:
        rating = rating + films[movies[row[0]]['title']][item]
    rating = rating/len(films[movies[row[0]]['title']])
    MovieLense.append([rating,len(films[movies[row[0]]['title']]),int(row[0])])
    IMDB.append([row[1],int(row[2].replace(',','')),int(row[0])])

MovieLense.sort(key=lambda x:(x[0],x[1],x[2]), reverse=True)
IMDB.sort(key=lambda x:(x[0],x[1]), reverse=True)

for i in range(len(lines)):
    MovieLense[i]=[MovieLense[i][2],i]
    IMDB[i]=[IMDB[i][2],i]

MovieLense.sort()
IMDB.sort()

f = open('rankContrast.csv', 'w', encoding='utf-8')
```

```
f.write('ID\tMovieLense Rank\tIMDB Rank\n')
for i in range(len(lines)):
    f.write('{}'.format(IMDB[i][0])+'\t{}'.format(MovieLense[i][1])+'\t{}'.format(IMDB[i][1])+'\n'
f.close()
```

Listing 7: The content of Q5.R

```
data=read.csv("rankContrast.csv", header=T, sep='\t')
plot(data[,"MovieLense.Rank"], data[,"IMDB.Rank"], xlab="MovieLense Rank",ylab="IMDB Rank")
cor.test(~MovieLense.Rank + IMDB.Rank, data=data, method="pearson")
```

Results:

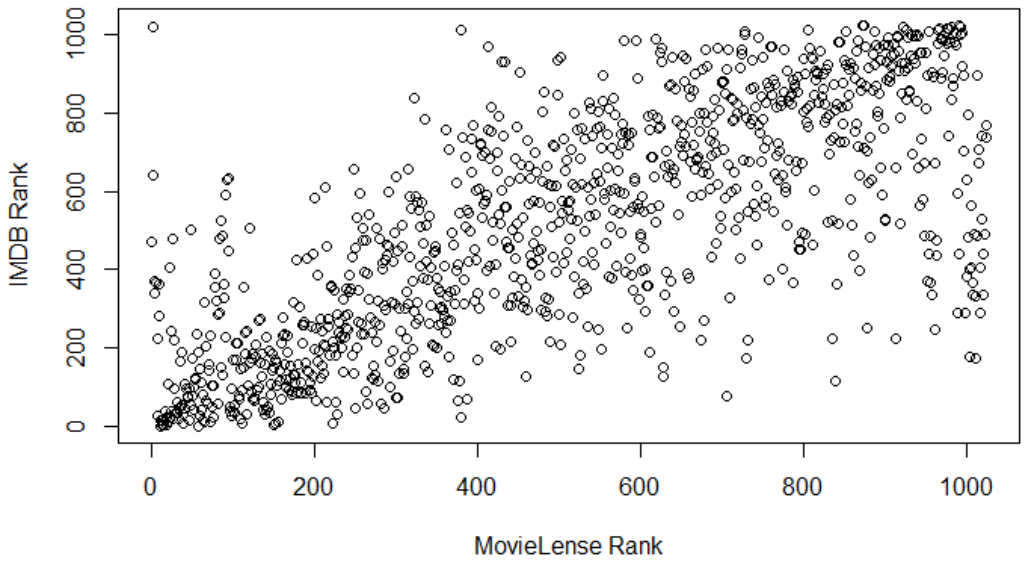

Figure 1: MovieLense Rank v.s. IMDB Rank

Table 6: Pearson's Correlation Test Result of MovieLense Rank v.s. IMDB Rank

Correlation	p-value
0.7456695	2.2e-16

According to the p-value, the correlation between these 2 ranks definitely exists. And the correlation index is 0.75, which tells that the correlation is relatively strong. So it may imply that the two user bases are interchangable.

6 MovieLense data v.s. IMDB Memento data

Repeat #5, but IMDB data from approximately July 31, 2005. What is the cumulative error from the desired target day of July 31, 2005?

Algorithm:

1. Use the script from "Programming Collective Intelligence" to open the movie data.
2. For each movie, open the URL and get the final URL from HTTP response.
3. Open the timemap of the final URL and get the memento list.
4. Iterate the memento list to get the first memento after 2005-07-31. Try to open this memento to get the rating and voters and save them into "IMDBmementoRating.csv". Otherwise, try next memento.
5. Open "IMDBmementoRating.csv" and use the script from "Programming Collective Intelligence" to open the MovieLense rating data and transform it.
6. For each movie in "IMDBmementoRating.csv", compute the voters and average rating of users in MovieLense data.
7. Sort the IMDB memento rating data and MovieLense rating data by rating and voters.
8. Replace the IMDB memento rating data and MovieLense rating data with movie ID and rank.
9. For each movie in "IMDBmementoRating.csv", save its ID, MovieLense rank and IMDB memento rank into "MementoRankContrast.csv".
10. Open "MementoRankContrast.csv" in R, plot the scatter and run the Pearson's correlation test.

Source code:

Listing 8: The content of mementoDownload.py

```
import urllib.request
from bs4 import BeautifulSoup
import json
import re
import datetime

def loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def getRating(url):
    with urllib.request.urlopen(url) as res:
        html = res.read()
```

```
    soup = BeautifulSoup(html, "html.parser")
    for b in soup.find_all('b'):
        m=re.search(r'.*(?=/10)', str(b.string))
        if m!=None:
            rating=m.group()
            break

    m=re.search(r'(?<=[(])\d.*(?=\ votes)', str(soup))
    voters=m.group()
    return rating, voters

def getMemento(link):
    with urllib.request.urlopen('http://memgator.cs.odu.edu/timemap/json/'+link) as res:
        html = res.read()
    soup = BeautifulSoup(html, "html.parser")
    timemap = json.loads(soup.string)
    for memento in timemap['mementos']['list']:
        m=re.search(r'.*(?=T)', memento['datetime'])
        if m.group()>='2005-07-31':
            row=[]
            row.append((datetime.datetime.strptime(m.group(), '%Y-%m-%d')-datetime.datetime(2005, 7
            (rating, voters)=getRating(memento['uri'])
            row.append(rating)
            row.append(voters)
            break
    return row


movies=loadMovies()
for i in range(len(movies)):
    print (i+1)
    try:
        with urllib.request.urlopen(movies['%d' %(i+1)]['URL']) as res:
            line=getMemento(res.geturl())
        f = open('IMDBmementoRating.csv','a',encoding='utf-8')
        f.write('{}'.format(i+1)+'\t'+line[1]+'\t'+line[2]+'\t{}'.format(line[0])+'\n')
        f.close()
    except:
        continue
```

Listing 9: The content of mementoMissingDownload.py

```
import urllib.request
from bs4 import BeautifulSoup
import json
import re
```

```
import datetime

def loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def getRating(url):
    with urllib.request.urlopen(url) as res:
        html = res.read()
    soup = BeautifulSoup(html, "html.parser")
    m=None
    for b in soup.find_all('b'):
        m=re.search(r'.*(?=/10)', str(b.string))
        if m!=None:
            rating=m.group()
            break

    if m==None:
        m=re.search(r'\d.*(?=/10)', str(soup))
        if m!=None:
            m=re.search(r'\d[.]\d', m.group())
            if m!=None:
                rating=m.group()
            else:
                rating='no'
        else:
            rating='no'

    m=None
    for a in soup.find_all('a'):
        if a.get('href')=='ratings':
            m=re.search(r'\d.*(?=␣votes)', str(a.string))
            if m!=None:
                voters=m.group()
                break
```

```
    if m==None:
        m=re.search(r'(?<=[(])\d.*(?=\ votes)', str(soup))
        if m!=None:
            voters=m.group()
        else:
            m=re.search(r'\d+(,\d{3})*(?=\ user\ ratings)', str(soup))
            if m!=None:
                voters=m.group()
            else:
                voters='no'

    return rating, voters

def getMemento(link):
    with urllib.request.urlopen('http://memgator.cs.odu.edu/timemap/json/'+link) as res:
        html = res.read()
    soup = BeautifulSoup(html, "html.parser")
    timemap = json.loads(soup.string)
    for memento in timemap['mementos']['list']:
        m=re.search(r'.*(?=T)', memento['datetime'])
        if m.group()>='2005-07-31':
            row=[]
            row.append((datetime.datetime.strptime(m.group(),'%Y-%m-%d')-datetime.datetime(2005, 7
            try:
                (rating, voters)=getRating(memento['uri'])
            except:
                rating='no'
                voters='no'
            if rating=='no' or voters=='no':
                continue
            else:
                row.append(rating)
                row.append(voters)
                break
    return row


f = open('IMDBrating.csv', 'r', encoding='utf-8')
lines = f.readlines()
f.close()
IMDB=[]
for line in lines:
    IMDB.append(int(line.split()[0]))

f = open('IMDBmementoRating.csv', 'r', encoding='utf-8')
lines = f.readlines()
```

```
f.close()

IMDBmemento=[]
for line in lines:
    IMDBmemento.append(int(line.split()[0]))

movies=loadMovies()
for i in IMDB:
    if i not in IMDBmemento:
        print (i)
        try:
            with urllib.request.urlopen(movies['%d' %(i)]['URL']) as res:
                line=getMemento(res.geturl())
            f = open('IMDBmementoRating.csv','a',encoding='utf-8')
            f.write('{}'.format(i)+'\t'+line[1]+'\t'+line[2]+'\t{}'.format(line[0])+'\n')
            f.close()
        except:
            continue
```

Listing 10: The content of Q6.py

```
def transformPrefs(prefs):
    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 loadMovies(path='ml-100k'):
    movies={}
    f = open(path+'/u.item', 'r', encoding='ISO-8859-1')
    lines = f.readlines()
    f.close()
    for line in lines:
        line=line.split('|')
        (id,title)=line[0:2]
        URL=line[4]
        movies.setdefault(id,{})
        movies[id]['title']=title
        movies[id]['URL']=URL
    return movies

def loadMovieLens(movies,path='ml-100k'):
```

```
    # Load data
    prefs={}
    for line in open(path+'/u.data', 'r', encoding='ISO-8859-1'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]['title']]=float(rating)
    return prefs

movies = loadMovies()
prefs = loadMovieLens(movies)
films = transformPrefs(prefs)

f = open('IMDBmementoRating.csv', 'r', encoding='UTF-8')
lines = f.readlines()
f.close()
MovieLense=[]
IMDB=[]
cumulativeError=[]

for line in lines:
    row = line.strip().split('\t')
    rating = 0
    for item in films[movies[row[0]]['title']]:
        rating = rating + films[movies[row[0]]['title']][item]
    rating = rating/len(films[movies[row[0]]['title']])
    MovieLense.append([rating,len(films[movies[row[0]]['title']]),int(row[0])])
    IMDB.append([row[1],int(row[2].replace(',','')),int(row[0])])
    cumulativeError.append([movies[row[0]]['title'],row[3]])

f = open('cumulativeError.txt','w',encoding='utf-8')
f.write('Film')
for i in range(61):
    f.write('␣')
f.write('Cumulative␣Error\n')
for line in cumulativeError:
    f.write(line[0])
    for i in range(65-len(line[0])):
        f.write('␣')
    f.write(line[1]+'\n')
f.close()

MovieLense.sort(key=lambda x:(x[0],x[1],x[2]), reverse=True)
IMDB.sort(key=lambda x:(x[0],x[1]), reverse=True)

for i in range(len(lines)):
    MovieLense[i]=[MovieLense[i][2],i]
```

```
    IMDB[i]=[IMDB[i][2],i]

MovieLense.sort()
IMDB.sort()

f = open('MementoRankContrast.csv', 'w', encoding='utf-8')
f.write('ID\tMovieLense Rank\tIMDB Rank\n')
for i in range(len(lines)):
    f.write('{}'.format(IMDB[i][0])+'\t{}'.format(MovieLense[i][1])+'\t{}'.format(IMDB[i][1])+'\n'
f.close()
```

Listing 11: The content of Q6.R

```
data=read.csv("MementoRankContrast.csv", header=T, sep='\t')
plot(data[,"MovieLense.Rank"], data[,"IMDB.Rank"], xlab="MovieLense Rank",ylab="IMDB Rank")
cor.test(~MovieLense.Rank + IMDB.Rank, data=data, method="pearson")
```

Results:

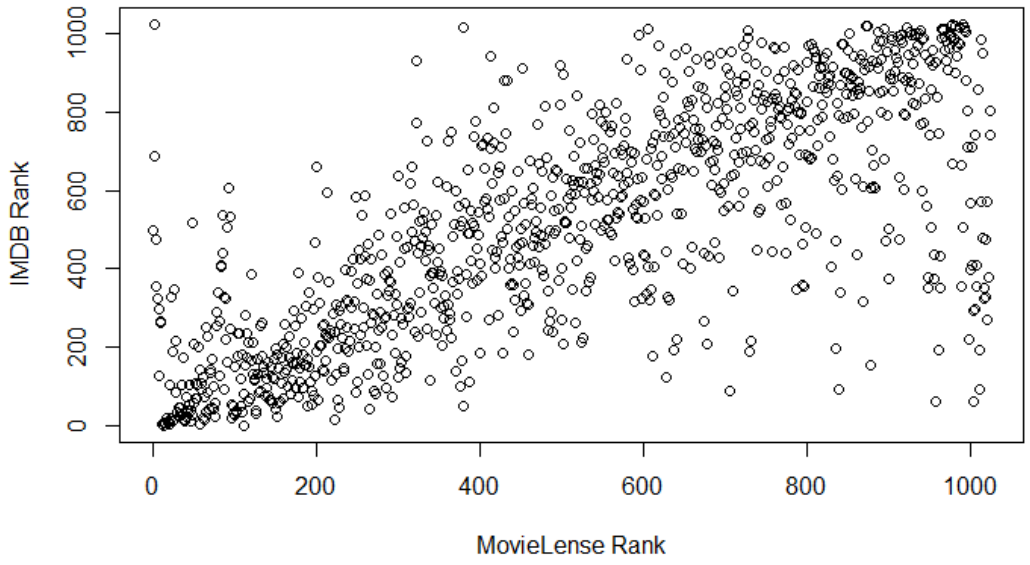

Figure 2: MovieLense Rank v.s. IMDB memento Rank

Table 7: Pearson's Correlation Test Result of MovieLense Rank v.s. IMDB Memento Rank

Correlation	p-value
0.755944	2.2e-16

Cumulative Error: see cumulativeError.txt

According to the p-value, the correlation between these 2 ranks definitely exists. And the correlation index is 0.76, which tells that the correlation is relatively strong. So it may imply that the two user bases are interchangable.