Web Science: Assignment #6

Alexander Nwala

Apurva Modi

Sunday, March 31, 2019

Contents

Problem 1

The goal of this project is to use the basic recommendation principles we have learned for user-collected data.
You will modify the code given to you which performs movie recommendations from the MovieLense data sets.
(https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter2/recommendations.py)
The MovieLense data sets were collected by the GroupLens Research Project at the University of Minnesota during the seven-month period from September 19th, 1997 through April 22nd, 1998.
We are using the "100k dataset", available for download from:
http://grouplens.org/datasets/movielens/100k/

There are three files which we will use:

1. u.data: 100,000 ratings by 943 users on 1,682 movies. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered.
2. u.item: Information about the 1,682 movies.
3. u.user: Demographic information about the users.

Find 3 users who are closest to you in terms of age, gender, and occupation. For each of those 3 users:

1. what are their top 3 favorite films?
2. bottom 3 least favorite films?

Based on the movie values in those 6 tables (3 users X (favorite + least)), choose a user that you feel is most like you. Feel free to note any outliers (e.g., "I mostly identify with user 123, except I did not like “Ghost” at all").
This user is the "substitute you".

SOLUTION :
Using. following steps:

1. Assigned Variables , 'Age', 'Gender' and 'Occupation' as '23', 'M', 'Student' respectively.
2. Selected the matching users **u.user** dataset, i received 6 such users.
3. Selected 3 users with user ID's : **135, 33, 706**

Listing 1: Assignment6_1.py

```
from operator import itemgetter
matchingUsers = []
apurvAge = 23
apurvOccupation = 'student'
apurvGender = 'M'
userMoviesDict = {}
userMovieRatingDict = {}
finalTopThree = {}
finalBottomThree = {}
userMovieRatingsList = []
movieRatingsList = []
matches = ''
bottomCount = 0
topCount = 0
listSize = 0

with open('u.user', 'r') as f1:
     for line in f1:
          userId,age,gender,occupation,zipcode = line.split('|')
          # if((int(age) < int(apurvAge) and int(age) > int((apurvAge - 3))) and (gender == apurvGend
          if((int(age) == apurvAge) and (gender == apurvGender) and (occupation == apurvOccupation))
               matchingUsers.append(userId)

print (matchingUsers)

with open('u.data', 'r') as f2:
     for line in f2:
          userId,movieId,rating,mseconds = line.split('     ')
          if(userId in matchingUsers):
               if(userId in userMoviesDict):
                    userMoviesDict[userId] = userMoviesDict[userId] + ":" + movieId + "|" + rating
               else :
                    userMoviesDict[userId] = movieId + "|" + rating

print('--------')
for key, value in userMoviesDict.items():
     # print(key,userMoviesDict[key])
     userMovieRatingsList = userMoviesDict[key].split(":")
     for movieRating in userMovieRatingsList:
          movie,rating = movieRating.split("|")
          userMovieRatingDict[movie] = rating
          # print(movie,rating)
```

```
    sortedRatings = sorted(userMovieRatingDict.items(), key=lambda value: value[1])
    # print("Length :",len(sortedRatings))
    bottomCount = 0
    topCount = 0
    listSize = 0
    bottomMovieData = ""
    topMovieData = ""
    for data in sortedRatings:
        listSize = listSize + 1
        if(bottomCount < 3):
            if(bottomMovieData == ""):
                bottomMovieData = str(data)
            else :
                bottomMovieData = bottomMovieData + ":" + str(data)
            bottomCount = bottomCount + 1
        if(listSize > len(sortedRatings) - 3):
            if(topMovieData == ""):
                topMovieData = str(data)
            else :
                topMovieData = topMovieData + ":" + str(data)

    finalBottomThree[key] = bottomMovieData
    finalTopThree[key] = topMovieData
    print('--------------')
    print(finalTopThree)
    print(finalBottomThree)
    print('\n')
print (" ******* TOP FAVORITE  MOVIES ******* ")
print ("User" + "  " + "Movie Title" + "  " + "Rating")
print ("----" + "  " + "-----------" + "  " + "------")
for key, value in finalTopThree.items():
    movieTuple = finalTopThree[key].split(":")
    for movie in movieTuple:
        movieId,rating = str(movie).split(",")
        movieId = movieId.replace("(","").replace("'","")
        with open('u.item', 'r') as file:
            for line in file:
                mid,movieTitle = line.split("|")[0:2]
                if(mid == movieId):
                    print (key,"  "+ movieTitle+"  "+rating.replace(")","").replace("'",""))

print('\n')

print (" ******* LEAST FAVORITE MOVIES ******* ")
print ("User" + "  " + "Movie Title" + "  " + "Rating")
print ("----" + "  " + "-----------" + "  " + "------")
for key, value in finalBottomThree.items():
    movieTuple = finalBottomThree[key].split(":")
    for movie in movieTuple:
        movieId,rating = str(movie).split(",")
        movieId = movieId.replace("(","").replace("'","")
        with open('u.item', 'r') as file:
            for line in file:
```

Problem 1 continued on next page. . .

```
                    mid,movieTitle = line.split("|")[0:2]
                    if(mid == movieId):
                        print (key,"   "+ movieTitle+"   "+
                            rating.replace(")","").replace("'",""))
```

The above code, will generate top favorites and least favorites movie from the selected 3 users.

Top 3 Favorite Movie		
User	Movies	Rating
135	Trainspotting (1996)	4
135	Liar Liar (1997)	4
135	Silence of the Lambs, The (1991)	5
33	Rosewood (1997)	4
33	Titanic (1997)	5
33	Silence of the Lambs, The (1991)	5
706	Titanic (1997)	5
706	Edge, The (1997)	5
706	Silence of the Lambs, The (1991)	5

Bottom 3 Favorite Movie		
User	Movies	Rating
135	Tales from the Hood (1995)	1
135	Jaws 2 (1978)	2
135	Assassins (1995)	2
33	Tales from the Hood (1995)	1
33	Jaws 2 (1978)	2
33	Assassins (1995)	2
706	Fargo (1996)	1
706	Game, The (1997)	1
706	Tales from the Hood (1995)	1

Figure 1: Top 3 and Bottom 3 Favorite Movies

Problem 2

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

SOLUTION

I selected **33** as '**Substitute Me** and pass the preferences of the 'Substitute Me' to the **sim_pearson** function, to determine the nearest 5 users. The following code has been rewrited and was originally taken from **Programming Collective Intelligence**

Listing 2: Assignment6_2.py

```
import csv
import math
import operator
import string
from collections import Counter
from math import sqrt

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 + sqrt(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)
```

Problem 2 continued on next page...

```
    # 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 topMatches(
    prefs,
    person,
    n=5,
    similarity=sim_pearson,
):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''

    scores = [(similarity(prefs, person, other), other) for other in prefs
              if other != person]
    scores.sort()
    scores.reverse()
    return scores[0:n]


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
```

Problem 2 continued on next page. . .

```python
                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 calculateSimilarItems(prefs, n=10):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''

    result = {}
    # Invert the preference matrix to be item-centric
    itemPrefs = transformPrefs(prefs)
    c = 0
    for item in itemPrefs:
        # Status updates for large datasets
        c += 1
        if c % 100 == 0:
            print('%d / %d' % (c, len(itemPrefs)))
        # Find the most similar items to this one
        scores = topMatches(itemPrefs, item, n=n, similarity=sim_distance)
        result[item] = scores
    return result
```

```python
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 loadMovieLens():
  # Get movie titles
     movies = {}
     for line in open('u.item'):
          (id, title) = line.split('|')[0:2]
          movies[id] = title
  # Load data
     prefs = {}
     for line in open('u.data'):
          (user, movieid, rating, ts) = line.split('\t')
          prefs.setdefault(user, {})
          prefs[user][movies[movieid]] = float(rating)
     return prefs

prefs = loadMovieLens()

with open('u.user') as tsv:
     for line in csv.reader(tsv, delimiter="|"):
        p2 = (line[0])
        p1 = '33'
        r = sim_pearson(prefs, p1, p2)
        with open('corrlate.csv','a') as f:
               writer=csv.writer(f)
               writer.writerow([r,p2,p1])
```

The above code will generate **corrlate.csv**, which gives the correlation of all the users in comparison with 'Substitute Me' i.e., user **33**

Negative Corelation		
Substitute Me	User	Corelation
33	900	-1
33	189	-0.981980506
33	483	-0.866025404
33	696	-0.866025404
33	17	-0.755928946

Positive Corelation		
Substitute Me	User	Corelation
33	305	0.92717265
33	457	0.944911183
33	288	0.948683298
33	679	0.970725343
33	127	1

Figure 2: Negative and Positive Correlation

Problem 3

3. 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 (i.e., films the substitute you is almost certain to hate).

SOLUTION

I made use of the **getRecommendations** function to get the recommendations for '**Substitute Me**. and the results of the same is saved in to a text file **recommendedMovies.txt** The following code has been rewrited and was originally taken from **Programming Collective Intelligence**

Listing 3: Assignment6_3.py

```python

import csv
import math
import operator
import string
from collections import Counter
from math import sqrt

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 + sqrt(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 topMatches(prefs, person, n=5, similarity=sim_pearson,):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''

    scores = [(similarity(prefs, person, other), other) for other in prefs
                if other != person]
    scores.sort()
    scores.reverse()
    return scores[0:n]


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)
```

Problem 3 continued on next page. . .

```
                    # 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 calculateSimilarItems(prefs, n=10):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''

    result = {}
    # Invert the preference matrix to be item-centric
    itemPrefs = transformPrefs(prefs)
    c = 0
    for item in itemPrefs:
        # Status updates for large datasets
        c += 1
        if c % 100 == 0:
            print('%d / %d' % (c, len(itemPrefs)))
        # Find the most similar items to this one
        scores = topMatches(itemPrefs, item, n=n, similarity=sim_distance)
        result[item] = scores
    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 loadMovieLens():
  # Get movie titles
    movies = {}
    for line in open('u.item'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open('u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
        print prefs[user][movies[movieid]]
    return prefs

prefs = loadMovieLens()
userId = '33'
r = getRecommendations(prefs, userId)
f = open("recommendedMovies.txt","w")
f.write(str(r))
f.close()
```

The above code will generate recommendations for **Substitute Me** in saves in to text file **recommended-Movies.txt**.

Top 5 Recommendations
Star Kid (1997)
Two or Three Things I Know About Her (1966)
Tough and Deadly (1995)
Santa with Muscles (1996)
Saint of Fort Washington, The (1993)

Bottom 5 Recommendations
3 Ninjas: High Noon At Mega Mountain (1998)
American Strays (1996)
Amityville 1992: It's About Time (1992)
Amityville 3-D (1983)
Amityville Curse, The (1990)

Figure 3: Top 5 and Bottom 5 Recommended Movies

Problem 4

4. Choose your (the real you, not the substitute you) favorite and least favorite film from the data. For each film, generate a list of the top 5 most correlated and bottom 5 least correlated films.Based on your knowledge of the resulting films, do you agree with the results? In other words, do you personally like / dislike the resulting films?

SOLUTION

transformPrefs function was changed to get the preferences and to get the top 5 suggestions from the **topMatches** function. The results for my favorite movie **The Shawshank Redemption** and my least favorite movie of that time **Jurassic Park** has been determined with positive and negative correlations. The following code has been rewrited and was originally taken from **Programming Collective Intelligence**

Listing 4: Assignment6_4.py

```
import csv
import math
import operator
import string
from collections import Counter
from math import sqrt

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 + sqrt(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 topMatches(
    prefs,
    person,
    n=5,
    similarity=sim_pearson,
):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''

    scores = [(similarity(prefs, person, other), other) for other in prefs
                if other != person]
    scores.sort()
    # scores.reverse()
    # return scores[0:n]
    lessfavorite = scores[:n]
    favorite = scores[-n:]
    return (lessfavorite, favorite)


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 calculateSimilarItems(prefs, n=10):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''

    result = {}
    # Invert the preference matrix to be item-centric
    itemPrefs = transformPrefs(prefs)
    c = 0
    for item in itemPrefs:
        # Status updates for large datasets
        c += 1
```

```
            if c % 100 == 0:
                 print('%d / %d' % (c, len(itemPrefs)))
            # Find the most similar items to this one
            scores = topMatches(itemPrefs, item, n=n, similarity=sim_distance)
            result[item] = scores
     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 loadMovieLens():
  # Get movie titles
     movies = {}
     for line in open('u.item'):
          (id, title) = line.split('|')[0:2]
          movies[id] = title
  # Load data
     prefs = {}
     for line in open('u.data'):
          (user, movieid, rating, ts) = line.split('\t')
          prefs.setdefault(user, {})
          prefs[user][movies[movieid]] = float(rating)
     return prefs

prefs = loadMovieLens()
prefs = transformPrefs(prefs)
(less, high) = topMatches(prefs, 'Shawshank Redemption, The (1994)')
f = open("moviePositiveCorrelation.txt","w")
f.write(str(less))
```

```
f.write('\n')
f.write(str(high))

(less, high) = topMatches(prefs, 'Jurassic Park (1993)')
f = open("moviepNegativeCorrelation.txt","w")
f.write(str(less))
f.write('\n')
f.write(str(high))
```

The program generates top 5 and bottom 5 recommendations for my favorite and least favorite movies and then they are saved in to **moviePositiveCorrelation.txt** and **moviepNegativeCorrelation.txt** text files respectively.

Top most Favorite Recommendations	
Movies	Corelation
Penny Serenade (1941)	1.000000003
Newton Boys, The (1998)	1.000000033
Oscar & Lucinda (1997)	1.000000003
Wedding Gift, The (1994)	1
Search for One-eye Jimmy, The (1996)	1

Top worst/least Favorite Recommendations	
Movies	Corelation
Loch Ness (1995)	1.00000004
Traveller (1997)	1
Traveller (1997)	1
Vermin (1998)	1
Wedding Gift, The (1994)	1

Bottom most Favorite Recommendations	
Movies	Corelation
Clean Slate (Coup de Torchon) (1981)	-1.00000004
1-900 (1994)	-1
American Dream (1990)	-1
Collectionneuse, La (1967)	-1
Colonel Chabert, Le (1994)	-1

Bottom worst/least Favorite Recommendations	
Movies	Corelation
Kaspar Hauser (1993)	-1.00000018
1-900 (1994)	-1
American Dream (1990)	-1
Anna (1996)	-1
Aparajito (1956)	-1

Figure 4: Top and Bottom most and worst/least Favorite Recommendations respectively

Conclusion

I am not in any position to make a comment on the result of the correlation because I am not sure about most of the movies here as I have not watched them, but I have watched **The Shawshank Redemption** and is one of the very best movie to watch., but the recommended movies based on this movie are obscure to me.

References

1. https://github.com/arthur-e/Programming-Collective-Intelligence

2. http://grouplens.org/datasets/movielens/100k/