Web Science: Homework 6
Recommendation Systems

Dr.Michele Weigle

Harveen Kaur

Thursday,March 31,2020

Contents

Problem 1

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?
- what are their bottom 3 (least favorite) films?

Based on the movie values in those 6 tables (3 users X (favorite + least favorite)), 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"). You can investigate more than just the top 3 and bottom 3 movies to find your best match.
This user is the substitute you.

SOLUTION :

I followed the following steps:

1. I have assigned the variables age, gender and occupation to be 23,F and student respectively.
2. Using the given dataset, I got 3 matches.
3. The matches were user 49,159 and 477.

Listing 1: Assignment6_1.py

```
from operator import itemgetter
matchingUsers = []
myage = 23
myoccupation = 'student'
mygender = 'F'
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(myage) and int(age) > int((myage - 3))) and (gender
          == mygender) and (occupation == myoccupation)):
          if((int(age) == myage) and (gender      == mygender) and (occupation
          == myoccupation)):
               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:
                mid,movieTitle = line.split("|")[0:2]
                if(mid == movieId):
                    print (key,"  "+ movieTitle+"  "+ rating.replace(")","").
                    replace("'",""))
```

The above given code generates the top favorite and least favorite movies from the selected users.

```

 ******* TOP FAVORITE  MOVIES *******
User  Movie Title  Rating
----  -----------  ------
49    Rosencrantz and Guildenstern Are Dead (1990)   5
49    Shallow Grave (1994)   5
49    Monty Pythons Life of Brian (1979)   5
159   Eraser (1996)   5
159   Mr. Hollands Opus (1995)   5
159   Feeling Minnesota (1996)   5
477   Nine Months (1995)   5
477   Sense and Sensibility (1995)   5
477   While You Were Sleeping (1995)   5

 ******* LEAST FAVORITE MOVIES *******
User  Movie Title  Rating
----  -----------  ------
49   Crow, The (1994)   1
49   Net, The (1995)   1
49   Ghost and the Darkness, The (1996)   1
159   Crow, The (1994)   1
159   Net, The (1995)   1
159   Ghost and the Darkness, The (1996)   1
477   Crow, The (1994)   1
477   Net, The (1995)   1
```

```
477   Ghost and the Darkness, The (1996)   1
```

Above given are the top favorite movies and least favorite movies. Screenshot of the terminal is available in the repo.

Problem 2

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

SOLUTION

I selected user 477 as the substitute me. Then passed the preference of the substitute me to the sim_-pearson function to determine the 5 most correlated users.

Listing 2: Assignment6_2.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)
                    # 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()

with open('u.user') as tsv:
     for line in csv.reader(tsv, delimiter="|"):
          p2 = (line[0])
```

```
        p1 = '477'
        r = sim_pearson(prefs, p1, p2)
        with open('correlate1.csv','a') as f:
            writer=csv.writer(f)
            writer.writerow([r,p2,p1])
```

The above code will generate the file correlate1.csv and will give the correlation of all the users and user 477(Substitute me).

```

 *******  NEGATIVE CORRELATION *******
User  Substitute Me  Correlation
----  -------------  -----------
677     477         -0.970725343
626     477             -1
752     477             -1
811     477             -1
856     477             -1

 ******* POSITIVE CORRELATION *******
User  Substitue ME  Correlation
----  ------------  -----------
 250    477         0.944911183
 321    477              1
 10     477              1
 67     477              1
 205    477              1
```

Above given is the positive and negative correlation.The data is saved in file **correlate1**.

Problem 3

Compute ratings for all the films that the substitute you has 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

```
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)
                    # 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 = '477'
r = getRecommendations(prefs, userId)
f = open("recommendedMovies.txt","w")
f.write(str(r))
f.close()
```

The above code will generate the file recommendedMovies and will give the top 5 and bottom 5 recommendations for 477(Substitute me).

```
***********TOP 5 RECOMMENDATIONS**************
(5.0, Wedding Gift, The (1994))
(5.0, Someone Elses America (1995))
(5.0, Some Mothers Son (1996))
(5.0, Santa with Muscles (1996))
(5.0, Saint of Fort Washington, The (1993))


***********BOTTOM 5 RECOMMENDATIONS ******************
(1.0, August (1996))
(1.0, Amityville 3-D (1983))
(1.0, American Strays (1996))
(1.0, 3 Ninjas: High Noon At Mega Mountain (1998))
(1.0, 1-900 (1994))
```

Above given are the top 5 and bottom 5 recommendations.

Problem 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

Here, **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 **Nine Months** and my least favorite movie of that time **Striking Distance** 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_1.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
```

Problem 4 continued on next page. . .

```
                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, 'Nine Months (1995)')
f = open("moviePositiveCorrelation.txt","w")
f.write(str(less))
f.write('\n')
f.write(str(high))

(less, high) = topMatches(prefs, 'Striking Distance (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 5 FAVORITE RECOMMENDATIONS**************
  Correlation         Movies
  -----------         ------
 1.0000000000000007  Daytrippers, The (1996)
 1.0000000000000007  Eves Bayou (1997)
 1.0000000000000007  Free Willy 3: The Rescue (1997)
 1.0000000000000007  Garden of Finzi-Contini, The (Giardino dei Finzi-Contini, Il(1970)
 1.0000000000000013  Microcosmos: Le peuple de lherbe (1996)


***********BOTTOM 5 FAVORITE RECOMMENDATIONS*****************
  Correlation         Movies
  -----------         ------
-1.0000000000000022   Steel (1997)
   -1.0               Anne Frank Remembered (1995)
   -1.0               Bloodsport 2 (1995)
   -1.0               Calendar Girl (1993)
```

```
   -1.0                Female Perversions (1996)
```

```


***********TOP 5 LEAST FAVORITE RECOMMENDATIONS**************
  Correlation         Movies
  -----------         ------
     1.0             Screamers (1995)
     1.0             Simple Twist of Fate, A (1994)
     1.0             Sleeper (1973)
     1.0             Timecop (1994)
1.0000000000000018   Paper, The (1994)


***********BOTTOM 5 LEAST FAVORITE RECOMMENDATIONS*****************
  Correlation         Movies
  -----------         ------
-1.0000000000000027  Bride of Frankenstein (1935)
-1.000000000000001   Boogie Nights (1997))
-1.000000000000001   Leaving Las Vegas (1995)
-1.0000000000000007  Flirting With Disaster (1996)
-1.0000000000000007  Kiss the Girls (1997)
```

Above given are the top 5 and least 5 favorite recommendations.
I chose **Nine Months** as my favorite movie. It was the only movie that I have seen. I haven't seen any of the recommended(top and least favorite) movies. Therefore, I cannot agree or disagree with the result.