Web Science: Assignment #6

Alexander Nwala

Puneeth Bikkasandra

Sunday, March 31, 2018

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 :

I have solved the problem as described in the below steps :

1. I have set the parameters, 'Age', 'Gender' and 'Occupation' as '27', 'M', 'Student' respectively.
2. Identified the matching users from **u.user** dataset, i received 6 such users.
3. Chose 3 users with following Ids : **758, 429, 104**

Listing 1: approximateUser.py

```
from operator import itemgetter

matchingUsers = []
myAge = 27
myOccupation = 'student'
myGender = 'M'
userMoviesDict = {}
userMovieRatingDict = {}
finalTopThree = {}
finalBottomThree = {}
userMovieRatingsList = []
movieRatingsList = []
matches = ''
bottomCount = 0
topCount = 0
listSize = 0

with open('users.txt', '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 (oc
        if((int(age) == myAge) and (gender == myGender) and (occupation == myOccupation)):
          matchingUsers.append(userId)

print matchingUsers

with open('data.txt', '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 "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('item.txt', '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 "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('item.txt', 'r') as file:
            for line in file:
```

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

The above code, will generate top 3 favorite and bottom 3 least favorite movies from the selected 3 users.

USER	MOVIE	RATING
758	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)	5
758	Trainspotting (1996)	5
758	Vertigo (1958)	5
429	Casablanca (1942)	5
429	Tombstone (1993)	5
429	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)	5
104	Casablanca (1942)	5
104	Tombstone (1993)	5
104	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)	5

Figure 1: Top 3 Favorite Movies

USER	MOVIE	RATING
758	Saint, The (1997)	1
758	Jackal, The (1997)	1
758	Conspiracy Theory (1997)	1
429	Amityville II: The Possession (1982)	1
429	Saint, The (1997)	1
429	Homeward Bound: The Incredible Journey (1993)	1
104	Starship Troopers (1997)	1
104	Con Air (1997)	1
104	Trees Lounge (1996)	1

Figure 2: 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

To solve this problem, i have used the code from the text **Programming Collective Intelligence** I have chosen user **429** as '**Substitute Me** and pass the preferences of the 'Substitute Me' to the **sim_pearson** function, to determine the nearest 5 users.

Listing 2: correlation.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
```

Problem 2 continued on next page. . .

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

prefs = loadMovieLens()

with open('users.txt') as tsv:
     for line in csv.reader(tsv, delimiter="|"):
        p2 = (line[0])
        p1 = '429'
        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 **429**

CORRELATION	USER	Substitute Me
-0.003776158	372	429
-0.001392455	586	429
-0.003747698	924	429
-0.00287144	940	429
-0.092554029	390	429

Figure 3: Negative Correlation

CORRELATION	USER	Substitute Me
0.942809042	813	429
0.9258201	926	429
0.866025404	842	429
0.862068966	443	429
0.87038828	675	429

Figure 4: 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

To solve this problem, i have used the code from the text **Programming Collective Intelligence** I have used the **getRecommendations** function to get the recommendations for '**Substitute Me**. The results of the same is saved in to a text file **recommendedMovies.txt**

Listing 3: recommendation.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('item.txt'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open('data.txt'):
        (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 = '429'
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
Boys, Les (1997)
They Made Me a Criminal (1939)
Star Kid (1997)
Someone Else's America (1995)
Santa with Muscles (1996)

Figure 5: Top 5 Recommended Movies

Bottom 5 Recommendations
August (1996)
Amityville: Dollhouse (1996)
Amityville: A New Generation (1993)'
3 Ninjas: High Noon At Mega Mountain (1998)
Amityville 1992: It's About Time (1992)

Figure 6: 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

To solve this problem, i have used the code from the text **Programming Collective Intelligence** I have used the **transformPrefs** function to change the preferences and get the top 5 suggestions from the **topMatches** function. The results for my favorite movie **Star Wars** and my least favorite movie **Jurassic Park** has been determined with positive and negative correlations

Listing 4: movieCorrelation.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]
        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
```

```python
        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('item.txt'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open('data.txt'):
        (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, 'Star Wars (1977)')
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 above code will generate top 5 and bottom 5 recommendations for my favorite and least favorite movies and are saved in to **moviePositiveCorrelation.txt** and **moviepNegativeCorrelation.txt** text files.

Correlation	Movie
1.0	Traveller (1997)
1.0	Two Much (1996)
1.0	Vermin (1998)
1.0	The Wedding Gift (1994)
1.0	Loch Ness (1995)

Figure 7: Top 5 Least Favorite Recommendations

Corrrelation	Movie
-1.0000000000000018	Kaspar Hauser (1993)
-1.0	1-900 (1994)
-1.0	American Dream (1990)
-1.0	Anna (1996)
-1.0	Aparajito (1956)

Figure 8: Bottom 5 Least Favorite Recommendations

Correlation	Movie
-1.000000000000004	Roseanna's Grave (For Roseanna) (1997)
-1.0000000000000018	Year of the Horse (1997)
-1.0000000000000007	I Like It Like That (1994)
-1.0	American Dream (1990)
-1.0	Der Bewegte Mann

Figure 9: Bottom 5 Favorite Recommendations

Correlation	Movie
1.0000000000000007	Hollow Reed (1996)
1.000000000000001	Designated Mourner, The (1997)
1.0000000000000013	Commandments (1997)
1.0000000000000013	Escape (1994)
1.000000000000004	Cosi (1996)

Figure 10: Top 5 Favorite Recommendations

References

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

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