INTRO. TO WEB SCIENCE: CS 532: A7

Due on Thursday, April 6, 2017

Dr. Nelson

Udochukwu Nweke

Contents

Problem 1 **3**

Problem 2 **13**

Problem 3 **13**

Problem 4 **14**

Problem 1

Listing 1: MovieLens Solutions Code

```
import math
from recommendations import *


def euclideanDistance(P, Q):

     if( len(P) != len(Q) ):
          return -1

     sumOfSquares = 0
     for i in range(0, len(P)):
          sumOfSquares += (P[i] - Q[i]) * (P[i] - Q[i])

     return math.sqrt(sumOfSquares)

def getUserDict(path='./data/movielens'):

     users = {}
     occupationMapping = {}
     for line in open(path + '/u.user'):
          (userid, age, gender, occupation, zipcode) = line.split('|')

          userid = userid.strip()
          age = age.strip()
          gender = gender.strip()
          occupation = occupation.strip()
          zipcode = zipcode.strip()

          transformedGender = 1
          if( gender == 'F' ):
               transformedGender = 0

          transformedOccupation = -1
          if( occupation in occupationMapping ):
               transformedOccupation = occupationMapping[occupation]
          else:
               occupationMapping[occupation] = len(occupationMapping)

          users.setdefault(userid, {})
          users[userid] = {'userid': userid, 'age': int(age), 'gender': gender,
        'transformed-gender':
        transformedGender, 'occupation': occupation, 'transformed-occupation':
        transformedOccupation, 'zipcode': zipcode}

     return users

def getClosestTriple(users):

     closestTriple = []

```

Problem 1 continued on next page. . .

```
    for userid in users:

        age = users[userid]['age']
        gender = users[userid]['transformed-gender']
        occupation = users[userid]['transformed-occupation']

        users[userid]['distance'] = euclideanDistance([age, gender, occupation],
        [32, 0, 5])

    sortedKeys = sorted(users, key=lambda userid:users[userid]['distance'])

    for i in range(0, 3):
        userid = sortedKeys[i]
        closestTriple.append( users[userid] )


    return closestTriple

def question1(users, prefs):

    '''
    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?
    - bottom 3 least favorite films?
    '''

    #1 part A:  Find 3 users who are closest to you in terms of age,
    # gender, and occupation.  For each of those 3 users:
    closestTriple = getClosestTriple(users)

    #350, 560, 890
    #1 part B: what are their top 3 favorite films?
    ##udoDict = {'transformed-occupation': 5, 'age': 32, 'transformed-gender':
    #0, 'zipcode': '23508', 'gender': 'F', 'occupation': 'student'}
    for user in closestTriple:
        print('user:', user)
        userRatedMovies = prefs[ user['userid'] ]

        topRatedMovies = sorted(userRatedMovies, key=lambda moviename:
        userRatedMovies[moviename])

        print('top 3 rated movies:')
        for i in range(len(topRatedMovies)-1, len(topRatedMovies)-4, -1):
            movie = topRatedMovies[i]
            print('\t', i+1, 'movie:', movie, ', rating:', userRatedMovies[movie])

        counter = 3
        if( len(topRatedMovies) < 3 ):
            counter = len(topRatedMovies)
```

Problem 1 continued on next page. . .

```
        print('bottom 3 rated movies:')
        for i in range(0, counter):
            movie = topRatedMovies[i]
            print('\t', i+1, 'movie:', movie, ', rating:', userRatedMovies[movie])


        print()

def question2(userid, prefs):

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

    scores = topMatches(prefs, userid, reverseFlag=True)
    print('\n5 Which 5 users are most correlated to user:', userid)
    for scoreUserTuple in scores:
        print('user:', scoreUserTuple[1], ', score:', scoreUserTuple[0])

    scores = topMatches(prefs, userid)
    print('\n5 Which 5 users are least correlated to user:', userid)
    for scoreUserTuple in scores:

        print('user:', scoreUserTuple[1], ', score:', scoreUserTuple[0])

def question3(userid, prefs):

    '''
    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).
    '''

    #pref with movies userid has not seen
    print('top 5 recommendations for films that the substitute ' + userid +
    ' should see')
    ratings = getRecommendations(prefs, userid)
    for i in range(len(ratings)-1, len(ratings)-6, -1):
        print('\t', ratings[i] )

    print('the bottom 5 recommendations that the substitute ' + userid +
    ' should not see')
    for i in range(0, 5):
        print('\t', ratings[i] )

def question4(prefs):

    '''
```

Problem 1 continued on next page. . .

```
    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?
    '''
    print('most similar for all')
    similarityDict = calculateSimilarItems(prefs, n=5, reverseFlag=True)
    for item, scores in similarityDict.items():

        print(item)
        print(scores)
        print()

    print('*'*20)
    print('*'*20)

    print('least similar for all')
    similarityDict = calculateSimilarItems(prefs, n=5, reverseFlag=False)
    for item, scores in similarityDict.items():

        print(item)
        print(scores)
        print()




users = getUserDict()
prefs = loadMovieLens()

substituteMe = '350'
#question2(substituteMe, prefs)

#question3(substituteMe, prefs)
question4(prefs)

'''
'''
```

Listing 2: MovieLens Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import sqrt
# A dictionary of movie critics and their ratings of a small set of movies
critics = {
    'Lisa Rose': {
        'Lady in the Water': 2.5,
        'Snakes on a Plane': 3.5,
        'Just My Luck': 3.0,
        'Superman Returns': 3.5,
```

Problem 1 continued on next page. . .

```
        'You, Me and Dupree': 2.5,
        'The Night Listener': 3.0,
    },
    'Gene Seymour': {
        'Lady in the Water': 3.0,
        'Snakes on a Plane': 3.5,
        'Just My Luck': 1.5,
        'Superman Returns': 5.0,
        'The Night Listener': 3.0,
        'You, Me and Dupree': 3.5,
    },
    'Michael Phillips': {
        'Lady in the Water': 2.5,
        'Snakes on a Plane': 3.0,
        'Superman Returns': 3.5,
        'The Night Listener': 4.0,
    },
    'Claudia Puig': {
        'Snakes on a Plane': 3.5,
        'Just My Luck': 3.0,
        'The Night Listener': 4.5,
        'Superman Returns': 4.0,
        'You, Me and Dupree': 2.5,
    },
    'Mick LaSalle': {
        'Lady in the Water': 3.0,
        'Snakes on a Plane': 4.0,
        'Just My Luck': 2.0,
        'Superman Returns': 3.0,
        'The Night Listener': 3.0,
        'You, Me and Dupree': 2.0,
    },
    'Jack Matthews': {
        'Lady in the Water': 3.0,
        'Snakes on a Plane': 4.0,
        'The Night Listener': 3.0,
        'Superman Returns': 5.0,
        'You, Me and Dupree': 3.5,
    },
    'Toby': {'Snakes on a Plane': 4.5, 'You, Me and Dupree': 1.0,
             'Superman Returns': 4.0},
}


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

#*** added option to reverse: reverseFlag
def topMatches(prefs, person, n=5, similarity=sim_pearson, reverseFlag=False):
    '''
    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()

    if( reverseFlag ):
        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()
    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]
```

Problem 1 continued on next page. . .

```
    return result


def calculateSimilarItems(prefs, n=10, reverseFlag=True):
    '''
    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,
        reverseFlag=reverseFlag)
        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(path='./data/movielens'):
  # Get movie titles
```

Problem 1 continued on next page. . .

```
    movies = {}
    for line in open(path + '/u.item'):
        (id, title) = line.split('|')[0:2]

        id = id.strip()
        title = title.strip()

        movies[id] = title
  # Load data
    prefs = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')

        user = user.strip()
        movieid = movieid.strip()
        rating = rating.strip()
        ts = ts.strip()

        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
    return prefs
```

The goal of this project is to use the basic recommendation principles we have learned for user-collected data. You will modify the code given to you which performs movie recommendations from the MovieLense data sets.

The MovieLense data sets were collected by the GroupLens Research Project at the University of Minnesota during the seven-month period from September 19th, 1997 through April 22nd, 1998. 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. This is a tab separated list of

```
user id | item id | rating | timestamp
```

The time stamps are unix seconds since 1/1/1970 UTC.
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? - 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 1:

In order to find 3 users who are closest to me in terms of age, gender, and occupation: I read the user data into a dictionary, with the key as userid and values represented by a vector containing age, gender, and occupation.

I created a mapping of Gender: Male represented by 1, Female represented by 0 and occupation: the first occupation (Technician) represented by 0, second occupation (Other) represented by 1, third occupation

Problem 1 continued on next page. . .

(Writer) -2, student -5, etc.

I used *euclideanDistance()* in listing 1 to compute euclidean distance between my vector representing my age, gender and occupation [32, 0, 5] and each user, and *getClosestTriple()*(Listing 1.) to retrieve the users closest to me. The result is seen in Table 1

Table 1: My Closest Triple

User id	Age	Gender	Occupation	Euclidean Distance
560	32	Male	Student	1.0
350	32	Male	Student	1.0
890	32	Male	Student	1.0

I sorted based on their movie ratings and picked their top 3 ratings and bottom 3 ratings. The result is seen in Table 2 through Table 7.

Table 2: User '350' Top 3 Favorite Films

Item	Movie-Title	Rating
1	Gone with the Wind (1939)	5.0
2	Casablanca (1942)	5.0
3	Empire Strikes Back, The (1980)	5.0

Table 3: User '350' 3 Least Favorite Films

Item	Movie-Title	Rating
1	Hunt for Red October, The (1990)	2.0
2	M*A*S*H (1970)	2.0
3	Contact (1997)	3.0

Table 4: User '560' Top 3 Favorite Films

Item	Movie	Rating
1	Star Wars (1977)	5.0
2	Chinatown (1974)	5.0
3	: Alien (1979)	5.0

Table 5: User '560' 3 Least Favorite Films

Item	Movie-Title	Rating
1	Event Horizon (1997)	1.0
2	Kids in the Hall: Brain Candy (1996)	1.0
3	Bed of Roses (1996)	1.0

The substitute me will be user '350'.The only difference is that I didn't like M*A*S*H (1970) at all. I will probably rate it 1.0 if I had to.

Table 6: User '890' Top 3 Favorite Films

Item	Movie-Title	Rating
1	To Kill a Mockingbird (1962)	5.0
2	One Flew Over the Cuckoo's Nest (1975)	5.0
3	Empire Strikes Back, The (1980)	5.0

Table 7: User '890' 3 Least Favorite Films

Item	Movie-Title	Rating
1	Star Trek: The Motion Picture (1979)	1.0
2	Ref, The (1994)	1.0
3	Batman (1989)	1.0

Problem 2

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

Solution 2:

In order to compute the users that are least correlated to the substitute me, which is user '350', I modified *topMatches()* in listing 2 by adding option to reverse to get most correlated users. The *topMatches()* gives the least correlated users by default. The result is seen in Table 8 and Table 9

Table 8: 5 Users Most Correlated to User '350'

Item	User	Correlation Coefficient
1	544	1.0
2	939	1.0
3	915	1.0
4	904	1.0
5	888	1.0

Table 9: 5 Users Least Correlated to User '350'

Item	User	Correlation Coefficient
1	133	-1.0
2	166	-1.0
3	17	-1.0
4	172	-1.0
5	190	-1.0

Problem 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

Problem 3 continued on next page...

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

Solution 3:

In order to recommend movies to user 350, I used *getRecommendations()* in listing 2 to compute the most recommended movies and least recommended movies by taking into account the movies user 350 have not seen. Table 10 and Table 11 shows the top and least recommended movies for user 350 respectively.

Table 10: 5 Most Recommended Movies for User '350'

Item	Movie-Title	Rating
1	They Made Me a Criminal (1939)	5.0
2	The Deadly Cure (1996)	5.0
3	Someone Else's America (1995)	5.0
4	Santa with Muscles (1996)	5.0
5	Prefontaine (1997)	5.0

Table 11: 5 Least Recommendation Movies for User '350'

Item	Movie-Title	Rating
1	3 Ninjas: High Noon At Mega Mountain (1998)	1.0
2	Amityville 1992: It's About Time (1992)	1.0
3	Amityville: A New Generation (1993)	1.0
4	Amityville: Dollhouse (1996)	1.0
5	B*A*P*S (1997)	1.0

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 4:

I chose the Beautician and the Beast, The (1997) as my favorite film and Steal Big, Steal Little (1995) as my least favorite. This because I have seen both movies.
In order to generate a list of the top most correlated and bottom least correlated movies for my favorite and least favorite movies, I used *calculateSimilarItems()* in listing 2. The result is seen from Table 12 through Table 15

I agree with the resulting films in Table 12 and Table 13. Except for Twin Town in Table 12. I would have included it in least correlated movies in Table 13.
For my least favorite film, I totally agree with the resulting films in Table 14 and Table 15 .

Table 12: 5 Most Correlated Movies to ‘Beautician and the Beast’

Item	Movie-Title	Similarity Rate
1	Wild America (1997)	1.0
2	Welcome To Sarajevo (1997)	1.0
3	Warriors of Virtue (1997)’	1.0
4	Twisted (1996)	1.0
5	Twin Town (1997)	1.0

Table 13: 5 Least Correlated Movies To Beautician and the Beast

Item	Movie-Title	Similarity Rate
1	Til There Was You (1997))	0
2	8 Seconds (1994)	0
3	A Chef in Love (1996)	0
4	Above the Rim (1994)	0
5	Across the Sea of Time (1995)	0

Table 14: 5 Most Correlated Movies To Steal Big, Steal Little (1995)

Item	Movie-Title	Similarity Rate
1	Yankee Zulu (1994)	1.0
2	Wyatt Earp (1994)	1.0
3	Woman in Question, The (1950)	1.0
4	Withnail and I (1987)	1.0
5	Wishmaster (1997)	1.0

Table 15: 5 Least Correlated Movies To Steal Big, Steal Little (1995)

Item	Movie-Title	Similarity Rate
1	Til There Was You (1997)	0
2	87 (1997)	0
3	3 Ninjas: High Noon At Mega Mountain (1998)	0
4	39 Steps, The (1935)	0
5	8 Heads in a Duffel Bag (1997)	0

References

[1] Toby Segaran. Programming Collective Intelligence, 2007.