# CS532S19: Assignment #6

Due on Sunday, March 30, 2019

*Nwala, Alexander C*

**Giridharan Ganeshkumar**

## Question 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 choose a user that you feel is most like you.**

1. The first step is to identify the user closest in terms of age, gender, and occupation.
2. In order to do this we read through the u.user file and check against the defined values based on the if condition
3. The initial thought was to include values for myself on the u.user but after the discussion in the class we decided to just go with the simple conditional check as we did not want to include new records to the record set
4. The next step is to find the top and bottom 3 favorite movies for each of the user identifies as closest
5. In order to find the top movies, we use the getTopMoviesForTheUser method. The method sorts the movies by ratings and returns both the top and bottom 3 in the result set.
6. The getTopMoviesForTheUser method is called within the loop to get the top and bottom 3 for all the identified users.
7. Looking through the result I mostly identify myself to the user with user id 350. 350 is the substitute user.

```
================================================================================
Closest to me in terms of age, gender, and occupation with user id:  350
================================================================================
Top 3 favorite films for user  350 :
 [['Raiders of the Lost Ark (1981)', 5.0], ['Manchurian Candidate, The (1962)', 5.0], ['Wild Bunch, The (1969)', 5.
0]]
Bottom 3 least favorite films for user  350 :
 [['Starship Troopers (1997)', 3.0], ['M*A*S*H (1970)', 2.0], ['Hunt for Red October, The (1990)', 2.0]]
================================================================================
Closest to me in terms of age, gender, and occupation with user id:  560
================================================================================
Top 3 favorite films for user  560 :
 [['Contact (1997)', 5.0], ['Citizen Kane (1941)', 5.0], ['Chinatown (1974)', 5.0]]
Bottom 3 least favorite films for user  560 :
 [['Kids in the Hall: Brain Candy (1996)', 1.0], ['Event Horizon (1997)', 1.0], ['Bed of Roses (1996)', 1.0]]
================================================================================
Closest to me in terms of age, gender, and occupation with user id:  890
================================================================================
Top 3 favorite films for user  890 :
 [['Empire Strikes Back, The (1980)', 5.0], ['To Kill a Mockingbird (1962)', 5.0], ['2001: A Space Odyssey (1968)',
5.0]]
Bottom 3 least favorite films for user  890 :
 [['Star Trek: The Motion Picture (1979)', 1.0], ['Ref, The (1994)', 1.0], ['Batman (1989)', 1.0]]
==============================================================================================================
==============================================================================================================
I mostly identify with user with user id 350
==============================================================================================================
==============================================================================================================
```

## Question 2

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

1. In order to find the most correlated to the substitute you we call the topMatches with user parameter as 350 and n as 5 and similarity as sim pearson
2. In order to find the least correlated to the substitute you we call the topMatchesReversed with

user parameter as 350 and n as 5 and similarity as sim pearson

3. topMatchesReversed is a method modified from the original topMatches method but the order just reversed using the sort method.

```
[(1.000000000000004, '544'), (1.0, '939'), (1.0, '915'), (1.0, '904'), (1.0, '888')]
==========================================================================================================
[(-1.0000000000000004, '133'), (-1.0, '166'), (-1.0, '17'), (-1.0, '172'), (-1.0, '190')]
==========================================================================================================
```

## Question 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**

1. In order to Provide a list of the top 5 recommendations for films that the substitute you should see we call the getRecommendations with user parameter as 350 and similarity as sim pearson
2. The result is stored in topMoviesForGivenUser and we get the top 5 and bottom 5 from the set which will be the top and least recommendations for the user.

```
Top 5 Recommendations for user  350 :
 [(5.0, 'They Made Me a Criminal (1939)'), (5.0, 'The Deadly Cure (1996)'), (5.0, "Someone Else's America (1995)"),
(5.0, 'Santa with Muscles (1996)'), (5.0, 'Prefontaine (1997)')]
Bottom 5 Recommendations for user  350 :
 [(1.0, 'B*A*P*S (1997)'), (1.0, 'Amityville: Dollhouse (1996)'), (1.0, 'Amityville: A New Generation (1993)'), (1.0,
"Amityville 1992: It's About Time (1992)"), (1.0, '3 Ninjas: High Noon At Mega Mountain (1998)')]
```

## Question 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?**

1. To get the required, we modify the calculateSimilarItems and name it calculateSimilarItemsByMovies.
2. Inside the calculateSimilarItemsByMovies we call both the topMatches and topMatchesReversed both and we get the top 5 correlated and least correlated for all.
3. The results are read and looped through and checked for favorite film in my case is the - The Silence of the Lambs
4. The result included Witness, The Deadly Cure, Substance of Fire, The, Spanish Prisoner and Wonderland. All the movies does seem to be related, i have not watched all the in the results After reading the synopsis I might still watch some of the movies not all. The recommendation is based on the movie and not based on the user, so it is definitely close to the movie given. In order to get more accurate results it would be better to see how i rated the movies so that the recommendations are a little more accurate than just based on movies.
5. The results from the least favorite movie is also similar to what is explained above.

Question 4 continued on next page. . .

```
Favorite Movie : Silence of the Lambs, The (1991)
==========================================================================================================
[(1.0, 'Wonderland (1997)'), (1.0, 'Witness (1985)'), (1.0, 'The Deadly Cure (1996)'), (1.0, 'Substance of Fire, The
(1996)'), (1.0, 'Spanish Prisoner, The (1997)')]
[(0, 'American Strays (1996)'), (0, 'August (1996)'), (0, 'B. Monkey (1998)'), (0, 'Big Bang Theory, The (1994)'),
(0, 'Bird of Prey (1996)')]
Least Favorite Movie :Addams Family Values (1993)
==========================================================================================================
[(1.0, 'You So Crazy (1994)'), (1.0, 'World of Apu, The (Apur Sansar) (1959)'), (1.0, 'Wild America (1997)'), (1.0,
'When Night Is Falling (1995)'), (1.0, 'War Room, The (1993)')]
[(0, "'Til There Was You (1997)"), (0, '3 Ninjas: High Noon At Mega Mountain (1998)'), (0, 'A Chef in Love (1996)'),
(0, 'Afterglow (1997)'), (0, 'Aiqing wansui (1994)')]
```

Listing 1: Python Script

```
#!/usr/bin/env python
# coding: utf-8

# In[7]:


#!/usr/bin/python
# -*- coding: utf-8 -*-
from math import sqrt
import csv
# 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,
        '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,
```

```python
        '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])
```

Question 4 continued on next page. . .

```
    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 topMatchesReversed(
    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()
    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:
```

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

    result = {}
    resultReveresed = {}
    # 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)
        scoresReversed = topMatchesReversed(itemPrefs, item, n=n,
            similarity=sim_distance)
        result[item] = scores
        resultReveresed[item] = scoresReversed
    return result,resultReveresed

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 getRecommendedItemsByMovies(prefs, itemMatch):
    userRatings = prefs
    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]:
            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


arrayofpref = []
def loadMovieLens():
  # Get movie titles
    movies = {}
    for line in open("C:\\6\\u.item"):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open("C:\\6\\u.data"):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
    return prefs

#def getCurrentUserRatings(pref,userid):

def getTopMoviesForTheUSer(prefs, person):
    currentUserMovieRating = []
    for other in prefs:
    # Don't compare me to myself
        if other == person:
                for item in prefs[other]:
                    currentUserMovieRating.append([item,prefs[person][item
                        ]])

    currentUserMovieRating =  sorted(currentUserMovieRating, key = lambda x
        : float(x[1]), reverse = True)
    return currentUserMovieRating[:3], currentUserMovieRating[-3:]
```

Question 4 continued on next page...

```python
pref = loadMovieLens()

with open("C:\\6\\u.user") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter='|')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            line_count += 1
        else:
            if row[1] == "32" and  row[2] == "M" and row[3] == "student":
                #Find 3 users who are closest to you in terms of age, gender, and occupation.  For each of those 3 users:
                print("==================================================================================")
                print("Closest to me in terms of age, gender, and occupation with user id: " , row[0])
                print("==================================================================================")
                #what are their top 3 favorite films?
                currentUserMovieRatingTop3, currentUserMovieRatingBottom3 = getTopMoviesForTheUSer(pref, row[0])
                print("Top 3 favorite films for user ",row[0],":\n",currentUserMovieRatingTop3)
                #What are bottom 3 least favorite films?
                print("Bottom 3 least favorite films for user ",row[0],":\n",currentUserMovieRatingBottom3)


print("==================================================================================")
print("==================================================================================")
print("I mostly identify with user with user id 350")
print("==================================================================================")
print("==================================================================================")


#Which 5 users are most correlated to the substitute you?
print(topMatches(pref,"350",n=5,similarity=sim_pearson,))
print("==================================================================================")

#Which 5 users are least correlated (i.e., negative correlation)?
print(topMatchesReversed(pref,"350",n=5,similarity=sim_pearson,))
print("==================================================================================")
```

Question 4 continued on next page. . .

```
#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
topMoviesForGivenUser = getRecommendations(pref,"350",similarity=sim_pearson)
print("Top 5 Recommendations for user ",350,":\n", topMoviesForGivenUser[:5])
#What are bottom 3 least favorite films?
print("Bottom 5 Recommendations for user ",350,":\n",topMoviesForGivenUser[-5:])



#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?

#Favorite 98|Silence of the Lambs, The (1991)|01-Jan-1991||http://us.imdb.com/M/title-exact?Silence%20of%20the%20Lambs,%20The%20(1991)|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|1|0|0
#Least favorite 386|Addams Family Values (1993)|01-Jan-1993||http://us.imdb.com/M/title-exact?Addams%20Family%20Values%20(1993)|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0

itemsim, itemsimreversed = calculateSimilarItemsByMovies(pref,5)



print("Favorite Movie : Silence of the Lambs, The (1991)")
print("==============================================================================================")
for line in itemsim:
    if(line == "Silence of the Lambs, The (1991)"):
        print(itemsim[line])
for line in itemsimreversed:
    if(line == "Silence of the Lambs, The (1991)"):
        print(itemsimreversed[line])

print("Least Favorite Movie :Addams Family Values (1993)")
print("==============================================================================================")

for line in itemsim:
    if(line == "Addams Family Values (1993)"):
        print(itemsim[line])
for line in itemsimreversed:
    if(line == "Addams Family Values (1993)"):
        print(itemsimreversed[line])
```