Justin Schaffner (JASchaff)
CS532 — Spring 2017

Assignment 7

April 7, 2017

Problem 1: User Like You

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 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".

Using the functions described in Chapter 2 of Programming Collective Intelligence by Toby Segaran and the movie lense data set from `http://grouplens.org/datasets/movielens/100k/`, its fairly easy to build similar functions to perform the demographic analysis needed on the file u.user from the data set. For this purpose I added the functions loadDemographics() and compareDemographics() as shown in Listing 1. These functions allow the user to compare their personal data to the list of users so they can find a user similar to themselves. The top three results returned for User=(Age= 36, Gender=M, Occupation=Student) were not similar to my personal movie preferences, so I looked further and ended up selecting the fourth candidate, user number 778 shown in Listing 2. The top three movies for 778 are definitely great movies, though why anyone would give E.T. a 1.0 is beyond me. All of the candidates liked movies I liked, but also hated movies I liked. Maybe I just like a lot of movies. Even so, 778 hated the least number of movies I like, and was spot on about "Free Willy" and "Angels in the Outfield".

Listing 1: Q1 Added Functions

```
def loadDemographics(path='data/'):
    #get user data
    users={}
    for line in open(path +'u.user'):
        (id_, age, gender, occupation, zip_)=line.split('|')
        users[id_]={'age': age, 'gender': gender, 'occupation':
           occupation}

    return users

def compareDemographics(users, user):
        alike={}
        for key in users:
            score=0
            score+= 2*(10-abs(int(users[key]['age']) - int(user['
               age'])))
            if users[key]['gender'].lower()==user['gender'].lower()
               :
                score+=20
            else: score+=10
```

```
            if users[key]['occupation'].lower()==user['occupation']
                ].lower():
                 score+=20
            else: score+=10
            alike[key]=score
        return alike
```

Listing 2: Q1 Results

```
378
   Top 3:
   [('Madness of King George, The (1994)', 5.0), ('Evita (1996)', 5.0),
      ('Amadeus (1984)', 5.0)]
   Bottom 3:
   [("Pete's Dragon (1977)", 2.0), ('Down Periscope (1996)', 1.0), ('Batman
      & Robin (1997)', 1.0)]
742
   Top 3:
   [('Contact (1997)', 5.0), ('Fargo (1996)', 5.0), ('Postino, Il (1994)',
      5.0)]
   Bottom 3:
   [('Star Trek: First Contact (1996)', 2.0), ('Mystery Science Theater
      3000: The Movie (1996)', 1.0), ('Broken Arrow (1996)', 1.0)]
124
   Top 3:
   [('Seven (Se7en) (1995)', 5.0), ('Manon of the Spring (Manon des sources)
      (1986)', 5.0), ("Monty Python's Life of Brian (1979)", 5.0)]
   Bottom 3:
   [('Platoon (1986)', 2.0), ('Princess Bride, The (1987)', 2.0), ("It's a
      Wonderful Life (1946)", 1.0)]
778
   Top 3:
   [('Seven (Se7en) (1995)', 5.0), ('Clueless (1995)', 5.0), ('Clerks
      (1994)', 5.0)]
   Bottom 3:
   [('Free Willy 2: The Adventure Home (1995)', 1.0), ('E.T. the
      Extra-Terrestrial (1982)', 1.0), ('Angels in the Outfield (1994)',
      1.0)]
350
   Top 3:
   [('Raiders of the Lost Ark (1981)', 5.0), ('Manchurian Candidate, The
      (1962)', 5.0), ('Wild Bunch, The (1969)', 5.0)]
   Bottom 3:
   [('Starship Troopers (1997)', 3.0), ('M*A*S*H (1970)', 2.0), ('Hunt for
      Red October, The (1990)', 2.0)]
```

Problem 2: User Correlation

Question 2: Which 5 users are most correlated to the substitute you? Which 5 users are least correlated (i.e., negative correlation)?
For the second portion, I loaded the Movie Lens data using the function from P.C.I. and used the topMatches() function to get the five users with highest correlation for user 778. I then created a similar function bottomMatches() shown in Listing 3 to get the 5 users with lowest correlation. At this point, I adjusted the if statement in the sim_pearson() function from "if n==0" to "if n<20" as discussed in the class discussion, in order to rule out comparisons between people who didn't rate at least 20 of the same movies. The returned results are shown in Listing 4 with the first number in the tuple representing the correlation score and the second their user ID.

Listing 3: Q2 Added Functions

```
def bottomMatches(prefs, person, n=5, similarity=sim_pearson):
    scores=[(similarity(prefs, person, other), other)
            for other in prefs if other!=person]
    scores.sort(key=lambda x: x[0])
    return scores[:n]
```

Listing 4: Q2 Results

```
Top Matches:
   (0.5715421732034413, '833') (0.5599586956015293, '233')
      (0.5406636991108152, '177') (0.5242672632055715, '886')
      (0.5046573785920919, '709')

Bottom Matches:
   (-0.5054209902584178, '38') (-0.502666685579464, '629')
      (-0.393211866521974, '712') (-0.376249330156684, '263')
      (-0.36434476859500914, '298')
```

Problem 3: Recommendations

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 (i.e., films the substitute you is almost certain to hate).
For this part, the existing functions from P.C.I. were sufficient to complete the task. Using getRecommendations() and user 778 as a stand-in generates the recommendations shown in Listing 5. The recommendations aren't far off from what I used to get when Hulu had the Criterion Collection, and the movies to avoid are all movies I know I

don't care for, so user 778 seems to be a good stand-in.

Listing 5: Q3 Results

```
Top 5 Movies to Watch:
   [(5.0, 'Aparajito (1956)'), (5.0, 'Mondo (1996)'), (5.0, 'Schizopolis
      (1996)'), (5.0, 'Saint of Fort Washington, The (1993)'), (5.0,
      'Little City (1998)')]
Top 5 Movies Not to Watch:
   [(1.0, '3 Ninjas: High Noon At Mega Mountain (1998)'), (1.0, 'Amityville:
      Dollhouse (1996)'), (1.0, 'Wishmaster (1997)'), (1.0, 'Police Story
      4: Project S (Chao ji ji hua) (1993)'), (1.0, 'Gordy (1995)')]
```

Problem 4: Film Correlation

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?
For this task, the existing functions transformPrefs(), getRecommendations(), topMatches() and bottomMatches() can be used to get the results needed. Using prefs, which came from the loadMovieLens() function used earlier, the function transformPrefs() returns a dictionary where the users and their movie rankings have been swapped for movies and how they are ranked by users. This allows the two functions topMatches() and bottomMatches() to perform the same calculation, but with a movie title as the starting point rather than a user and their ranking data. The only annoying thing about the returned dictionary is that the ID has been removed, so you have to pass the exact movie title to the function. For my top film I chose "The Birdcage". Super cheese, but it makes me happy. For the worst film I went with "Santa with Muscles". It was definitely not my choice to watch this. One of my roommates in college was weirdly obsessed. The results from these two entries found in Listing 6 aren't exactly perfect as far as what I like. I'm not sure why "Amityville Horror" ranked so highly, unless its just based on campy old movies. The other recommendations are pretty good, though I've never seen "Mary Reilly". The non recommendations are also fairly legit, except for "The Englishman Who Went Up a Hill, But Came Down a Mountain". It was really slow and not exactly a favorite, but it wasn't terrible. "Santa with Muscles" actually returns no correlation to any movies, so the functions just return the beginning and the end of a list of zero correlations. According to the data, only two people rated it, and both of them gave it a 5 (WHY?!?). I changed the 'if' statement in sim_pearson() to n==0 just to be sure that wasn't messing it up and got the exact same result for "Santa with Muscles", although it did shift a lot of the other results for questions 1 through 3. The two users did rate other movies, so the 'if n==0' statement shouldn't be the one returning 0 in every case. That means the denominator must be 0 for other movies

that those two had rated. Since we're only comparing two individuals, its possible that this is the only film for which their tastes converge. Its also possible that their 5 ratings were the result of extensive drug use, or temporary loss of sanity. Those are the only scenarios I can come up with. Just to be thorough, I ran it again with "Kazaam", but it met a similar fate. "Ace Ventura" did get some actual results though, shown in Listing 7. Among the films with high correlation to "Ace Ventura" I did see "The Return of the Pink Panther" and liked it as a kid, but I doubt I'd watch it now. I haven't seen the others. The movies with low correlation includes "Breakfast at Tiffany's", definitely a classic. Our cat is named after the cat in the movie. His name is Cat.
The code for Recommendations.py can be found in Listing 8 and the code used to call the functions, Movie_Lens.py, in Listing 9.

Listing 6: Q4 First Results

```
Birdcage, The (1996) Top 5:
   (0.6634445232597436, 'Basketball Diaries, The (1995)')
      (0.6623350408559849, 'Gigi (1958)') (0.602333061800304, 'Amityville
      Horror, The (1979)'(0.5886209782368717, 'Midnight in the Garden of
      Good and Evil (1997)') (0.5839864791803113, 'Mary Reilly (1996)')
Birdcage, The (1996) Bottom 5:
   (-0.48538420461213694, "Nobody's Fool (1994)") (-0.4411024868807589,
      'Shadow, The (1994)') (-0.4107261562920037, 'Night on Earth (1991)')
      (-0.40001214425349346, 'Ice Storm, The (1997)') (-0.3915487484750877,
      'Englishman Who Went Up a Hill, But Came Down a Mountain, The (1995)')
Santa with Muscles (1996) Top 5:
   (0, 'Scream of Stone (Schrei aus Stein) (1991)') (0, 'You So Crazy
      (1994)') (0, "Mat' i syn (1997)") (0, 'B. Monkey (1998)') (0, 'Sweet
      Nothing (1995)')
Santa with Muscles (1996) Bottom 5:
   (0, 'Kolya (1996)') (0, 'Mrs. Doubtfire (1993)') (0, "Muriel's Wedding
      (1994)") (0, 'Shall We Dance? (1996)') (0, 'Stand by Me (1986)')
```

Listing 7: Q4 Second Results

```
Birdcage, The (1996) Top 5:
   (0.6634445232597436, 'Basketball Diaries, The (1995)')
      (0.6623350408559849, 'Gigi (1958)') (0.602333061800304, 'Amityville
      Horror, The (1979)'(0.5886209782368717, 'Midnight in the Garden of
      Good and Evil (1997)') (0.5839864791803113, 'Mary Reilly (1996)')
Birdcage, The (1996) Bottom 5:
   (-0.48538420461213694, "Nobody's Fool (1994)") (-0.4411024868807589,
      'Shadow, The (1994)') (-0.4107261562920037, 'Night on Earth (1991)')
      (-0.40001214425349346, 'Ice Storm, The (1997)') (-0.3915487484750877,
      'Englishman Who Went Up a Hill, But Came Down a Mountain, The (1995)')
Ace Ventura: Pet Detective (1994) Top 5:
```

```
   (0.6003962008842328, 'Return of the Pink Panther, The (1974)')
      (0.567535564335793, 'Escape from New York (1981)')
      (0.5662475270344743, 'Shine (1996)') (0.5561908729799682, 'Four Rooms
      (1995)') (0.5343906279692369, "City Slickers II: The Legend of
      Curly's Gold (1994)")
Ace Ventura: Pet Detective (1994) Bottom 5:
   (-0.47067872433164165, 'Somewhere in Time (1980)') (-0.4513367815043889,
      'Ref, The (1994)') (-0.429858597921356, 'Benny & Joon
      (1993)')(-0.3896940350361718, 'Home for the Holidays (1995)')
      (-0.38776813904538016, "Breakfast at Tiffany's (1961)")
```

Listing 8: Recommendations.py

```
from math import sqrt
from sys import stdout
import os
#import requests
#import bs4

# Returns a distance-based similarity score for person1 and person2


def sim_distance(prefs,person1,person2):
    # Get the list of shared_items
    si={}
    for item in prefs[person1]:
        if item in prefs[person2]:
            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[person1][item]-prefs[
                person2][item],2)
                                for item in prefs[person1] if item
                                    in prefs[person2]])
            return 1/(1+sum_of_squares)

def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item]=1
    # Find the number of elements
```

```
    n=len(si)

    # if they are no ratings in common, return 0
    if n<20: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up 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 up the products
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

    # Calculate 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

# Returns the best matches for person from the prefs dictionary.
# Number of results and similarity function are optional params.
def topMatches(prefs,person,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other),other)
            for other in prefs if other!=person]
    # Sort the list so the highest scores appear at the top
    scores.sort(key=lambda x: x[0])
    scores.reverse( )
    return scores[:n]

def bottomMatches(prefs, person, n=5, similarity=sim_pearson):
    scores=[(similarity(prefs, person, other), other)
            for other in prefs if other!=person]
    scores.sort(key=lambda x: x[0])
    return scores[:n]


# Gets recommendations for a person by using a weighted average
# of every other user's rankings
def getRecommendations(prefs,person,similarity=sim_pearson):
    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)
                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(key=lambda x: x[0])
    rankings.reverse( )
    return rankings


def transformPrefs(prefs):
    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 initializeUserDict(tag,count=5):
    user_dict={}
    # get the top count' popular posts
    for p1 in get_popular(tag=tag)[0:count]:
        # find all users who posted this
        for p2 in get_urlposts(p1['href']):
            user=p2['user']
```

```
            user_dict[user]={}
    return user_dict


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(path='data/'):
    # Get movie titles
    movies={}
    for line in open(path +'u.item', encoding='latin-1'):
        (id_,title)=line.split('|')[0:2]
        movies[id_]=title
    # Load data
    prefs={}
    for line in open(path +'u.data'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]]=float(rating)
    return prefs


def loadDemographics(path='data/'):
    #get user data
    users={}
    for line in open(path +'u.user'):
        (id_, age, gender, occupation, zip_)=line.split('|')
        users[id_]={'age': age, 'gender': gender, 'occupation':
           occupation}

    return users

def compareDemographics(users, user):
        alike={}
        for key in users:
            score=0
            score+= 2*(10-abs(int(users[key]['age']) - int(user['
               age'])))
            if users[key]['gender'].lower()==user['gender'].lower()
               :
                score+=20
            else: score+=10
            if users[key]['occupation'].lower()==user['occupation'
               ].lower():
                score+=20
            else: score+=10
            alike[key]=score
        return alike
```

```
def loadMovies(path='data/'):
    #get movie data
    movies={}
    for line in open(path+'u.item'):
        (id_,title,date,url)=line.split('|')[0:4]
        movies[title]=url
    return movies
```

Listing 9: Movie_Lens.py

```
from recommendations import *
import sys
import json

#question 1: collects user information for comparison
users=loadDemographics()
age=input('Enter age: ')
gender=input('Enter gender: ')
occupation=input('Enter most correct occupation from list {
   technician, writer, executive, administrator, student, lawyer,
   educator, scientist, entertainment, programmer, librarian,
   homemaker, engineer, artist, marketing, healthcare, doctor,
   retired, salesman, none, other}: ')
#asks for the number of returned results the user wants. I did this
    because the top 3 were not good matches
results=int(input('Enter number of results required: '))
user={'age': age, 'gender': gender, 'occupation':occupation}
alike=compareDemographics(users, user)
alike_sorted= sorted(alike.items(), key=lambda x: x[1], reverse=
   True)

m_alike=list(alike_sorted[x][0] for x in range(results))
print(*m_alike, sep='\t')

del users
del alike

prefs=loadMovieLens()
#prints likely candidates and their preferences
for u in m_alike:
    temp=sorted(prefs[u].items(), key=lambda x: x[1], reverse=True)
    print(u+'\n', 'Top 3:\n', temp[:3], '\n\tBottom 3:\n', temp
       [-3:], sep='\t')

del m_alike
```

```
#user selects candidate most like them in movie preferences
a_user=input('Select the user most like you: ')
#Question 2: finds top 5 and bottom 5 correlations to candidate
    selected
top=topMatches(prefs, a_user)
bottom=bottomMatches(prefs, a_user)
print('Top Matches:\n', *top, '\n', sep='\t')
print('Bottom Matches:\n', *bottom, '\n', sep='\t')
#Question 3: finds top 5 movie recommendations and top 5 movies not
    to watch
rankings=getRecommendations(prefs, a_user)
print('Top 5 Movies to Watch:\n', rankings[:5], '\nTop 5 Movies Not
    to Watch:\n', rankings[-5:], sep='\t')

del a_user, top, bottom, rankings
#Question 4: get matches from personal fav and hate from the list
fav=input('Enter your favorite Movie from the List: ')
hate=input('Enter your least favorite Movie from the List: ')

items=transformPrefs(prefs)
favtop=topMatches(items, fav)
favbottom=bottomMatches(items, fav)
hatetop=topMatches(items, hate)
hatebottom=bottomMatches(items, hate)
print(fav+' Top 5:\n', *favtop, '\n'+fav+' Bottom 5:\n', *favbottom
    , sep='\t')
print(hate+' Top 5:\n', *hatetop, '\n'+hate+' Bottom 5:\n', *
    hatebottom, sep='\t')
```