CS896 Introduction to Web Science
Fall 2013
Report for Assignment 8

Corren G. McCoy

November 14, 2013

Contents

List of Tables

1 Question 1

1.1 Problem

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. You are to modify recommendations.py to answer the following questions. Each question your program answers correctly will award you 10 points. You must have the question answered completely correct; partial credit will only be awarded if your answer is very close to the correct one.

1.2 Methodology

We used the Python source code for determining user and item similarity as found in Segaran, 2007[1]. Since many of the questions for this project are suitable for database queries, we incorporated the SQLite[1] library. The product's web site describes it as 'a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.' In addition to the provided core functionality, *recommendations.py*, performs the following tasks:

- Creates a SQLite database and tables for the MovieLens files (u.data, u.item, u.user);
- Loads the MovieLens data and populates the database tables using insert statements;
- Calculates user-based and item-based similarity using the Pearson R correlation coefficient; and
- Uses the user-based metrics to rank groups of similar and dissimilar users.

The modified source code is shown in Appendix A. For those particular responses which were obtained using an SQL query, we have noted the query statement in addition to the response. We applied a filter to ensure that high rankings obtained from the ratings of a single user were discarded. In cases where multiple records met the selection criteria, all of those records are shown in the response. It should be noted that while the expected range of values for the Pearson coefficient is +1 to -1, we did obtain correlation values greater than 1. This can most likely be attributed to Python's floating point arithmetic and internal precision. None of the Pearson scores presented in this report were truncated after calculation.

1.3 Query Responses

1. What 5 movies have the highest average ratings? Show the movies and their ratings sorted by their average ratings. (Table 1)

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem
where udata.movie_id = uitem.movie_id
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

2. What 5 movies received the most ratings? Show the movies and the number of ratings sorted by number of ratings. (Table 5)

Movie Id	Movie Title	Avg. Rating	No. Raters
1189	Prefontaine (1997)	5.0	3
1293	Star Kid (1997)	5.0	3
1467	Saint of Fort Washington, The (1993)	5.0	2
1500	Santa with Muscles (1996)	5.0	2
1449	Pather Panchali (1955)	4.625	8

Table 1: Highest Average Ratings

```
select uitem.movie_id,  movie_title, count(*)
from udata, uitem
where udata.movie_id = uitem.movie_id
group by uitem.movie_id, movie_title
order by 3 desc
limit 5;
```

Movie Id	Movie Title	No. Ratings
50	Star Wars (1977)	583
258	Contact (1997)	509
100	Fargo (1996)	508
181	Return of the Jedi (1983)	507
294	Liar Liar (1997	485

Table 2: Movies with Most Ratings

3. What 5 movies were rated the highest on average by women? Show the movies and their ratings sorted by ratings. (Table 3)

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='F'
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

Movie Id	Movie Title	Avg. Rating	No. Raters
50	Mina Tannenbaum (1994)	5.0	2
258	Schindler's List (1993)	4.633	79
100	Close Shave, A (1995)	4.632	19
181	Shawshank Redemption, The (1994)	4.563	64
294	Wallace & Gromit: The Best of Aardman Animation (1996)	4.533	15

Table 3: Highest on Average by Women

4. What 5 movies were rated the highest on average by men? Show the movies and their ratings sorted by ratings. (Table 4)

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='M'
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

Movie Id	Movie Title	Avg. Rating	No. Raters
1293	Star Kid (1997)	5.0	3
1175	Hugo Pool (1997)	5.0	2
1189	Prefontaine (1997)	5.0	2
1467	Saint of Fort Washington, The (1993)	5.0	2
1500	Santa with Muscles (1996)	5.0	2

Table 4: Highest on Average by Men

5. What movie received ratings most like Top Gun? Which movie received ratings that were least like Top Gun (negative correlation)?

```
recommendations.topMatches(movieprefs,"Top Gun (1986)",1)
[(1.0000000000000027, 'Shiloh (1997)')]

recommendations.worstMatches(movieprefs,"Top Gun (1986)",1)
[(-1.0000000000000007, 'Babysitter, The (1995)')]
```

6. Which 5 raters rated the most films? Show the raters' IDs and the number of films each rated. (Table 5)

```
select user_id, count(*) as count
from udata
group by user_id
order by 2 DESC
limit 5;
```

7. Which 5 raters most agreed with each other? Show the raters' IDs and Pearson's r, sorted by r. (Tables 6 and 7)

8. Which 5 raters most disagreed with each other (negative correlation)? Show the raters' IDs and Pearson's r, sorted by r. (Tables 8 and 9)

9. What movie was rated highest on average by men over 40? By men under 40? (Tables 10 and 11)

User ID	No. Films Rated
405	737
655	685
13	636
450	540
270	518

Table 5: Most Film Ratings

	Group 1	Group 2	Group 3
My User ID	772	135	139
User ID 1	889	810	879
Pearson	1.0000000000000047	1.000000000000004	1.000000000000004
User ID 2	899	79	610
Pearson	1.000000000000004	1.000000000000004	1.000000000000004
User ID 3	780	552	607
Pearson	1.000000000000004	1.000000000000004	1.000000000000004
User ID 4	277	351	278
Pearson	1.000000000000004	1.000000000000004	1.000000000000004
Cum. Diff	-1.66533453694e-14	-1.59872115546e-14	-1.59872115546e-14

Table 6: Most Similar Raters

	Group 4	Group 5
My User ID	170	191
User ID 1	764	733
Pearson	1.000000000000004	1.000000000000004
User ID 2	292	60
Pearson	1.000000000000004	1.000000000000004
User ID 3	257	517
Pearson	1.000000000000004	1.000000000000004
User ID 4	238	170
Pearson	1.000000000000004	1.000000000000004
Cum. Diff	-1.59872115546e-14	-1.59872115546e-14

Table 7: Most Similar Raters - cont'd

	Group 1	Group 2	Group 3
My User ID	928	86	832
User ID 1	547	756	622
Pearson	-1.000000000000004	-1.000000000000004	-1.000000000000004
User ID 2	432	630	613
Pearson	-1.000000000000004	-1.000000000000004	-1.000000000000004
User ID 3	317	251	491
Pearson	-1.000000000000004	-1.000000000000004	-1.000000000000004
User ID 4	112	196	267
Pearson	-1.000000000000004	-1.000000000000004	-1.000000000000004
Cum. Diff	8.0	8.0	8.0

Table 8: Most Dissimilar Raters

	Group 4	Group 5
My User ID	794	761
User ID 1	799	667
Pearson	-1.000000000000004	-1.000000000000004
User ID 2	740	640
Pearson	-1.000000000000004	-1.000000000000004
User ID 3	469	600
Pearson	-1.000000000000004	-1.000000000000004
User ID 4	129	384
Pearson	-1.000000000000004	-1.000000000000004
Cum. Diff	8.0	8.0

Table 9: Most Disimilar Raters - cont'd

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='M'
and age >= 40
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

Movie Id	Movie Title	Rating	No. Raters
1558	Aparajito (1956)	5.0	4
1512	World of Apu, The (Apur Sansar) (1959)	5.0	3
1293	Star Kid (1997)	5.0	2
1302	Late Bloomers (1996)	5.0	2
1449	Pather Panchali (1955)	4.8	5

Table 10: Highest Avg. Men Over 40

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='M'
and age < 40
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

10. What movie was rated highest on average by women over 40? By women under 40? (Tables 12 and 13)

Movie Id	Movie Title	Rating	No. Raters
1175	Hugo Pool (1997)	5.0	2
1467	Saint of Fort Washington, The (1993)	5.0	2
1500	Santa with Muscles (1996)	5.0	2
1167	Sum of Us, The (1994)	4.5	4
851	Two or Three Things I Know About Her (1966)	4.5	2

Table 11: Highest Avg. Men Under 40

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='F'
and age >= 40
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc, 4 desc
limit 5;
```

Movie Id	Movie Title	Rating	No. Raters
904	Ma vie en rose (My Life in Pink) (1997	5.0	3
169	Wrong Trousers, The (1993)	5.0	2
1203	Top Hat (1935)	5.0	2
1194	Once Were Warriors (1994)	4.8	5
241	Last of the Mohicans, The (1992)	4.667	3

Table 12: Highest Avg. Women Over 40

```
select uitem.movie_id,  movie_title, round(avg(rating),3), count(*)
from udata, uitem, uuser
where udata.movie_id = uitem.movie_id
and uuser.user_id = udata.user_id
and uuser.gender='F'
and age < 40
group by uitem.movie_id, movie_title
having count(*) > 1
order by 3 desc,4 desc
limit 5;
```

[1] http://sqlite.org/

Movie Id	Movie Title	Rating	No. Raters
113	Horseman on the Roof, The (Hussard sur le toit, Le) (1995)	5.0	2
1153	Backbeat (1993)	5.0	2
114	Wallace & Gromit: The Best of Aardman Animation (1996)	5.0	11
320	Paradise Lost: The Child Murders at Robin Hood Hills (1996)	4.819	5
1084	Anne Frank Remembered (1995)	4.8	5

Table 13: Highest Avg. Women Under 40

Bibliography

[1] T. Segaran. *Programming collective intelligence: building smart web 2.0 applications.* O'Reilly Media, 2007.

Appendix A

Python Source

```
from __future__ import division
from math import sqrt
import sqlite3

'''
@Author Corren McCoy, 2013
@Purpose
Item and user-based recommendations using logic
from textbook "Programming Collective Intelligence"
'''

# A dictionary of movie critics and their ratings of a small
# set of movies (sample dataset)
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, '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}}

conn = sqlite3.connect("mydatabase.db") # or use :memory: to put it in RAM
cursor = conn.cursor()

# 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]])
  ## Changed per errata for page 11
  # return 1/(1+sum_of_squares)
  return 1/(1+sqrt(sum_of_squares))

# Returns the Pearson correlation coefficient for p1 and p2
## Changed per errata corrected version found here:
## http://stackoverflow.com/a/13562198/1828663

# Returns the Pearson correlation coefficient for p1 and p2
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

  # if they are no ratings in common, return 0
  if len(si)==0: return 0

  ## Changed per errata page 13
  # Sum calculations
  n=float(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

# 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]
  scores.sort()
  scores.reverse()
  return scores[0:n]

## Returns the worst matches for person from the prefs dictionary.
## Number of results and similarity function are optional params.
def worstMatches(prefs,person,n=5,similarity=sim_pearson):
  scores=[(similarity(prefs,person,other),other)
                  for other in prefs if other!=person]
  scores.sort()
  return scores[0: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()
```

```
  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 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
    # Changed to use Pearson
    #scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
    scores=topMatches(itemPrefs,item,n=n,similarity=sim_pearson)
    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='C:/Python27/myFiles/Assignment 8/data/movielens', insert="N"):
  # Get movie titles
  movies={}
  for line in open(path+'/u.item'):
   (movieid,title)=line.split('|')[0:2]
   movies[movieid]=title
   mi=str(unicode(movieid)).encode('UTF-8')
   mt=str(unicode(title)).encode('UTF-8')
   if insert=="Y":
     # insert some data (movie_id, movie_title)
     cursor.execute("INSERT INTO uitem VALUES (?,?)", (mi, mt))


  # Load the ratings
  prefs={}
  for line in open(path+'/u.data'):
    (user,movieid,rating,ts)=line.split('\t')
    prefs.setdefault(user,{})
    prefs[user][movies[movieid]]=float(rating)
    if insert=="Y":
      # insert data (user_id, movie_id, rating)
      cursor.execute("INSERT INTO udata VALUES (?,?,?)", (user, movieid, float(rating)))

  # Load the user demographics
  users={}
  for line in open(path+'/u.user'):
    (user,age,gender,occupation, zip_code)=line.split('|')
    if insert=="Y":
      # insert data (user_id, age, gender)
      cursor.execute("INSERT INTO uuser VALUES (?,?,?)", (user, age, gender))


  # save data to database
  conn.commit()
  return prefs

## Maintain u.data in a database so we can use SQL queries
def openSQL():
  try:
```

```
    # create tables(s)
    cursor.execute("""CREATE TABLE udata
                   (
                   user_id integer, movie_id integer, rating integer
                    )
                """)
    # Catch the exception
  except Exception as e:
      print e, "Deleting all rows"
      cursor.execute("""DELETE from udata""")
      conn.commit()
  try:
    cursor.execute("""CREATE TABLE uitem
                   (
                   movie_id integer, movie_title text
                    )
                """)
  # Catch the exception
  except Exception as e:
     print e, "Deleting all rows"
     cursor.execute("""DELETE from uitem""")
     conn.commit()
  try:
     cursor.execute("""CREATE TABLE uuser
                   (
                   user_id integer, age integer, gender text)
                """)
  # Catch the exception
  except Exception as e:
    print e, "Deleting all rows"
    cursor.execute("""DELETE from uuser""")
    conn.commit()

def mostAgreed(userset):
  rankings={}
  for user, rating in userset.items():
    difference=0
    for value, key in rating[0:4]:
      # Difference between top four scores and me (4)
      difference= difference + (1-value)
    rankings[user]=difference
  # Return the top 5 rankings from smallest to highest
  count=0
  print "These user ratings most agreed"
  for key, value in sorted(rankings.iteritems(), key=lambda (k,v): (v,k)):
    if (count < 5):
      print key, userset[key][0:4], value
      #print "%s: %s" % (key, value)
    count = count + 1
```

```
  return rankings

def mostDisagreed(userset):
  rankings={}
  stop=len(userset)
  start=stop-5
  for user, rating in userset.items():
    difference=0
    for value, key in rating[start:stop]:
      # Difference between bottom four scores and me (4)
      difference= difference + (1-value)
    rankings[user]=difference
  # Return the top 5 rankings from smallest to highest
  count=0
  print "These user ratings most disagreed"
  for key, value in sorted(rankings.iteritems(), key=lambda (k,v): (v,k),reverse=True):
    if (count < 5):
      print key, userset[key][start:stop], value
      #print "%s: %s" % (key, value)
    count = count + 1
  return rankings
## Main driver added
if __name__ == "__main__":
  # Open database connection. Create tables.
  openSQL()

  # user-based similarity
  print "Calculating user-based similarity"
  userprefs=loadMovieLens(insert="Y")
  moviesim=calculateSimilarItems(userprefs,n=1682) # total number of movies
  print "Calculating item-based similarity"

  # item based similarity
  movieprefs=transformPrefs(userprefs)
  usersim=calculateSimilarItems(movieprefs,n=943) # total number of users

  # Which 5 most agreed
  usersA=mostAgreed(usersim)
  usersD=mostDisagreed(usersim)
```