CS-432/532 Introduction to Web Science:
Assignment #7:
User-Collected Data

Due on Thursday, March 31, 2016

Dr. Michael L. Nelson

Plinio Vargas
pvargas@cs.odu.edu

Contents

List of Figures

Listings

List of Tables

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.

   Example:

196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488

2. u.item: Information about the 1,682 movies. This is a tab separated list of

   movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation |Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |

   The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data data set.

   Example:

161|Top Gun (1986)|01-Jan-1986||http://us.imdb.com/M/title-exact?Top%20Gun%20(1986)|0|1|0|0|0|0|0|0|0|0|0|0|0|1|0|0|0|0
162|On Golden Pond (1981)|01-Jan-1981||http://us.imdb.com/M/title-exact?On%20Golden%20Pond%20(1981)|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0
163|Return of the Pink Panther, The (1974)|01-Jan-1974||http://us.imdb.com/M/title-exact?Return%20of%20the%20Pink%20Panther,%20The

3. u.user: Demographic information about the users. This is a tab separated list of:

   user id | age | gender | occupation | zip code

   The user ids are the ones used in the u.data data set.

Example:

1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213

The code for reading from the u.data and u.item files and creating recommendations is described in the book Programming Collective Intelligence. Feel free to modify the PCI code to answer the following questions.

Problem 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 ll").

This user is the "substitute you".

1.1 Approach

There were three approach considerations to answer the questions in this assignment:

a. Using R to upload, query and analyze the data.

b. Saving data as string, manipulating the data using python scripts.

c. Follow [1] advice and store data into a database for manipulation.

We followed the latest approach. PostgreSQL was selected as our DBMS. Various bash shells and python scripts were created to upload and manipulate the data from PostgreSQL. Very small modifications were made to the solutions available in [1] in order to be adapted into Python 3.4 syntax.

First, we created the database **cs752_db** using bash shell *CreateDb.sh*:

Listing 1: CreateDb.sh

```
#! /bin/bash
# CREATE DATABASE postgresql

echo
echo "Creating database cs532..."
psql -c "CREATE DATABASE cs532_db;"
```

```
7
8  echo
9  echo "Creating Extension unaccent ..."
10 psql cs532_db -c "CREATE EXTENSION IF NOT EXISTS unaccent;"
11
12 echo
13 echo "Creating Extension cube ..."
14 psql cs532_db -c "CREATE EXTENSION IF NOT EXISTS cube;"
```

The script creates the database and some extensions.

Second, we create the tables and populate them using a bash shell and python. The section below is just a portion of the script showing the syntax of creating user's table *user_tb* from the command line:

Listing 2: Creating Table Using CreatePopulateTbls.sh

```
21
22 echo
23 echo "Creating table user_tb ...."
24 psql cs532_db -c "CREATE TABLE IF NOT EXISTS user_tb ( user_id INT,
25                                                        age INT,
26                                                        gender VARCHAR,
27                                                        occupation VARCHAR,
28                                                        zipcode VARCHAR,
29                                                      PRIMARY KEY (user_id));"
```

Below is another portion of the same script, but showing how the table gets populated:

Listing 3: Loading Table Via CreatePopulateTbls.sh Script

```
61 ##
62 # INSERT or COPY data to postgresql table
63 echo
64 echo "Extracting data from ml-100k/u.user into user_tb"
65 psql cs532_db -c "\COPY user_tb ( user_id,
66                                   age,
67                                   gender,
68                                   occupation,
69                                   zipcode
70                                 )FROM PROGRAM 'python3 TransferUserData.py'"
```

The data set obtained from `http://grouplens.org/datasets/movielens/100k/` were uploaded into the corresponding **postgres** tables shown below:

u.data → rating_tb
u.item → movie_tb
t.user → user_tbl

Once the tables were created (line 1-30), the remaining portion of the script populates the data into the respective tables by using "COPY" instruction. This instruction is a display of the text files via *python*, which after given the data-fields character separators replace them with a tab for **postgres**. For example:

Listing 4: Example of Simple Data Upload: TransferUserData.py

Problem 1 continued on next page. . .

```
7  def main():
8      with open('ml-100k/u.user', mode='rt', encoding='iso-8859-1') as file:
9          for line in file:
10             record = line.replace('|', '\t').strip()
11             print(record)
12     return
```

Listing 4 is the simplest form of data upload. In order cases is a little more complex because we have to account for missing data fields or a misalignment for character separators in the data. This cannot be done in advance, it is done based on trial and error, patching the mistakes as the data gets uploaded into the tables.

Listing 5: Example of Complex Data Upload: TransferMoviesData.py

```
11     with open('ml-100k/u.item', mode='rt', encoding='iso-8859-1') as file:
12         for line in file:
13             record = line.split('|')
14             for i in range(len(record) - 19):
15                 counter += 1
16                 if record[i] == 'unknown':
17                     print(record[i], '\t01-Jan-1990\t ', '\t', end='')
18                 elif record[i].strip():
19                     print(record[i].strip(), '\t', end='')
20             if counter == 5 and not record[4].strip() and record[1] != 'unknown':
21                 print(' \t', end='')
22             for i in range(len(record) - 19, len(record)):
23                 genre.append(int(record[i].strip()))
24
25             print(tuple(genre), end='')
26             counter = 0
27             genre = []
28             print()
29     return
```

The example in Listing 5 is more complex than Listing 4. Listing 5 accounts for a non-uniform data. During data upload there was a movie titled *unknown* without a date field, so instead of discarding this record a data field was added (lines 16-17). In addition, the script is compressing 19 genres data-fields into a single data-field (lines 22-23).

1.2 Solution

1.2.1 3-Users Closest to Me in Terms of Age, Gender, and Occupation

To find 3 users who are closest to me in terms of age, gender, and occupation, we performed a search for the all the student males between the ages of 35 to 45 years old limiting to 3 tuples. That search yielded the following results:

```
select * from user_tb
where age >= 35 and age <= 45 and
occupation = 'student' and gender = 'M'
limit 3;
```

Table 1: 3-Users Closest Match

Id	Age	Gender	Occupation	Zip-Code
378	35	M	student	02859
188	42	M	student	29440
565	40	M	student	55422

Values were obtained from user_tb using a SQL query.

Problem 1 continued on next page. . .

1.2.2 3-Users Closest to Me Top 3 Favorite Films

To answer this question 3 queries were performed in **postgres**:

```
postgres@lg-server:~$ psql cs532_db
psql (9.3.11)
Type "help" for help.

cs532_db=# select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 565 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
 user_id |                title                | rating
---------+-------------------------------------+--------
     565 | Enchanted April (1991)              |      5
     565 | Madness of King George, The (1994)  |      5
     565 | Stonewall (1995)                    |      5
(3 rows)

cs532_db=# select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 188 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
 user_id |            title            | rating
---------+-----------------------------+--------
     188 | Princess Bride, The (1987)  |      5
     188 | Braveheart (1995)           |      5
     188 | Mary Poppins (1964)         |      5
(3 rows)

cs532_db=# select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 378 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
 user_id |            title            | rating
---------+-----------------------------+--------
     378 | Killing Fields, The (1984)  |      5
     378 | Victor/Victoria (1982)      |      5
     378 | Usual Suspects, The (1995)  |      5
(3 rows)

cs532_db=#
```

The query combined *rating_tb* with *movie_tb* sorted by the ranking field in a decedent fashion, limiting to only 3 tuples, and then extracting only the desired fields: **user_id**, movie **title** and **rating**.

Then, to obtain the highest ranking movies from user 188 we query:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 188 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
```

Yielding table:

Problem 1 continued on next page. . .

Table 2: 3-Higest Ranked Films by User 188

user_id	Title	Rating
188	Princess Bride, The (1987)	5
188	Braveheart (1995)	5
188	Mary Poppins (1964)	5

Values were obtained from combining ranking_tb and movie_tb

Then, to obtain the 3-highest movies ranked by user 378 we just change the user_id field to 378, leaving everything else the same:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 378 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
```

Yielding table:

Table 3: 3-Higest Ranked Films by User 378

user_id	Title	Rating
378	Killing Fields, The (1984)	5
378	Victor/Victoria (1982)	5
378	Usual Suspects, The (1995)	5

Values were obtained from combining ranking_tb and movie_tb

The same applies to user 565:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 565 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating desc
limit 3;
```

Yielding table:

Table 4: 3-Higest Ranked Fimls by User 565

user_id	Title	Rating
565	Enchanted April (1991)	5
565	Madness of King George, The (1994)	5
565	Stonewall (1995)	5

Values were obtained from combining ranking_tb and movie_tb

Problem 1 continued on next page. . .

1.2.3 3-Users Closest to Me Least 3 Favorite Films

Different from sub-section before, in order to obtain the least favorite ranked films from selected users the only change in our query is to order the ranking field from descending fashion in an ascending one. Then, to get the least favorite ranked movies by user 188 we can query **postgres** *cs752_db*:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 188 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating asc
limit 3;
```

Yielding table:

Table 5: 3-Lowest Ranked Films by User 188

user_id	Title	Rating
188	Beavis and Butt-head Do America (1996)	1
188	Congo (1995)	2
188	Bullets Over Broadway (1994)	2

Values were obtained from combining ranking_tb and movie_tb

For user 378:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 378 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating asc
limit 3;
```

Yielding table:

Table 6: 3-Lowest Ranked Films by User 378

user_id	Title	Rating
378	Batman & Robin (1997)	1
378	Down Periscope (1996)	1
378	She's the One (1996)	2

Values were obtained from combining ranking_tb and movie_tb

Finally, for user 565 we may query:

```
select r.user_id, m.title, r.rating
from rating_tb as r, movie_tb as m
where r.user_id = 565 and r.item_id = m.movie_id
group by r.user_id, m.title, r.rating
order by rating asc
limit 3;
```

Problem 1 continued on next page...

Yielding table:

Table 7: 3-Lowest Ranked Films by User 565

user_id	Title	Rating
565	Muriel's Wedding (1994)	2
565	Wings of Desire (1987)	3
565	Cook the Thief His Wife & Her Lover, The (1989)	4

Values were obtained from combining ranking_tb and movie_tb

1.2.4 Substitute Me

Selecting substitute me was a painful (**water-boarding**) decision. I could not identify with user 188, although I would rank "Braveheart (1995)" equally with a 5, "Princess Bride, The (1987)" and "Mary Poppins (1964)" would be around 2 instead of 5.

I am very distant from user 565 all the highest ranked films by 565 would be in my lowest ranked films selection.

I am not that close either with user 378, the highest ranked films will be in my 3 rate selection.

Solution:

```
User 378 was selected as substitute me.
```

Problem 2

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

2.1 Approach

To answer questions 2 through 4 a Python program was created (*GetCorr.py*) which output is display below:

```
/usr/bin/python3.4 "/DiskStation/CS-532 Webscience/a7/GetCorr.py"
Starting Time: Wed, Mar 30, 2016 at 17:33:16

Top 5 most correlated substitute me
+----------+----------------+
| Movie_id | Recomm Ranking |
+----------+----------------+
|   369    |     1.000      |
|   651    |     0.878      |
|   841    |     0.866      |
|   341    |     0.866      |
|   149    |     0.818      |
+----------+----------------+


Bottom 5 correlated substitute me
+----------+----------------+
| Movie_id | Recomm Ranking |
+----------+----------------+
|   866    |     -0.730     |
|   677    |     -0.681     |
|   723    |     -0.614     |
|   812    |     -0.603     |
|   471    |     -0.600     |
+----------+----------------+


            Substitute me 5 top unseen movies recommendations

+----------+-------------------------------------------------+----------------+
| Movie_id |             Title                               | Recomm Ranking |
+----------+-------------------------------------------------+----------------+
|   1189   | Prefontaine (1997)                              |     5.000      |
|   1653   | Entertaining Angels: The Dorothy Day Story (1996) | 5.000    |
|   1617   | Hugo Pool (1997)                                |     5.000      |
|   1599   | Someone Else\'s America (1995)                   |     5.000      |
|   1536   | Aiqing wansui (1994)                            |     5.000      |
+----------+-------------------------------------------------+----------------+



           Substitute me 5 least unseen movies recommendations
```

Problem 2 continued on next page...

```
+----------+-------------------------------------------------+----------------+
| Movie_id |             Title                               | Recomm Ranking |
+----------+-------------------------------------------------+----------------+
|   314    | 3 Ninjas: High Noon At Mega Mountain (1998)  |     1.000     |
|   437    | Amityville 1992: It's About Time (1992)      |     1.000     |
|   439    | Amityville: A New Generation (1993)          |     1.000     |
|   599    | Police Story 4: Project S (Chao ji ji hua) (1993) | 1.000    |
|   784    | Beyond Bedlam (1993)                         |     1.000     |
+----------+-------------------------------------------------+----------------+

End Time: Wed, Mar 30, 2016 at 17:33:17
Execution Time: 0.72 seconds

Process finished with exit code 0
```

We used *pysycopg2* python library to connect with the database and the *sim_pearson* function developed in [1]. The function expects the user preferences. **GetCorr.py** stores substitute me preferences in variable substitute_me (lines 33-36).

Listing 6: Finding Closest Substitute Me Match: GetCorr.py

```
33      # get my substitute me information
34      my_sql = getSQL()
35      my_index = 378
36      substitute_me = getPrefs(cursor, my_sql(my_index))
37      # print(substitute_me)
38
39      # iterate through all users
40      all_prefs = {}
41      for person in all_users:
42          other_index = person[0]
43          other_person = getPrefs(cursor, my_sql(other_index))
44
45          prefs = {other_index: other_person, my_index: substitute_me}
46          all_prefs[other_index] = other_person
47
48          # print(prefs)
49          #print(other_index, sim_pearson(prefs, other_index, my_index))
50
51          # build everyone preference
52          corr_table[other_index] = sim_pearson(prefs, other_index, my_index)
```

Substitute me preference information is entered into an iteration that compares with everyone else preference (lines 40 - 52). The correlation is calculated by passing **substitute me** preference with the other user preference to function *sim_pearson*. The result of each calculation is stored in a table *corr_table* in line 52. If we sort *corr_table* in ascending order by ranking, the bottom portion will correspond to the closest correlation to **substitute me** in terms of moving ranking criteria. The top portion of the table will be the opposite.

2.2 Solution

2.2.1 5-Users Most Correlated to the Substitute Me

Table 8: Top 5 Correlated Subustitute Me

Movie_id	Correlation
369	1.000
651	0.878
841	0.866
341	0.866
149	0.818

Values were obtained by running *GetCorr.py* and retrieving the 5 bottom values of table *corr_table*. The very last value is actually user 378 with a correlation value of 1. The very first row (user 369) has a correlation of 0.99999, but is rounded to 1.000 after formatting the data.

2.2.1 5-Users Least Correlated to the Substitute Me

Table 9: 5-Users Least Correlated to the Substitute Me

Movie_id	Correlation
866	-0.730
677	-0.681
723	-0.614
812	-0.603
471	-0.600

Values were obtained by running *GetCorr.py* and retrieving the 5 top values of table *corr_table* which corresponds with the elements with lower correlation index.

Problem 3

Compute rankings for all the films that the substitute you hasn't 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).?

3.1 Approach

The preference of all users was stored in variable *all_prefs* (line 46 in Listin 6), this information along with my user_id is passed to function *getRecommendations* obtain from [1]. The result is stored in variable *recommendations* which will contain the ranking of movies not yet seen by **substitute me** ordered from least most likely desired recommended preference to the most likely recommended preference.
Then, to answer the question we just take the top or bottom portion of the array.

3.2 Solution

3.2.1 Top 5 Recommendations of Unseen films for Substitute Me

Table 10: Top 5 Substitute Me Recommendations of Unseen films

Movie_id	Title	Rating
1189	Prefontaine (1997)	5.000
1653	Entertaining Angels: The Dorothy Day Story (1996)	5.000
1617	Hugo Pool (1997)	5.000
1599	Someone Else's America (1995)	5.000
1536	Aiqing wansui (1994)	5.000

Values were obtained bottom portion of the array recommendation from function *getRecommendations*

3.2.2 Bottom 5 Recommendations of Unseen films for Substitute Me

Table 11: Bottom 5 Recommendations of Unseen films for Substitute Me

Movie_id	Title	Rating
314	3 Ninjas: High Noon At Mega Mountain (1998)	1.000
437	Amityville 1992: It's About Time (1992)	1.000
439	Amityville: A New Generation (1993)	1.000
599	Police Story 4: Project S (Chao ji ji hua) (1993)	1.000
784	Beyond Bedlam (1993)	1.000

Values were obtained top portion of the array recommendation from function *getRecommendations*

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?

4.1 Solution

The recommendations results are not surprising to me. As stated in **1.2.4 Substitute Me** solution user 378 is closest to me by default. However, the results are close in terms how I feel about **substitute me** ranking. I dislike all movies not recommended and the recommended ones I would rate them between 2 and 3, similar to the highest ranking for user 378.

Problem 5

Rank the 1,682 movies according to the 1997/1998 MovieLense data. Now rank the same 1,682 movies according to todays (March 2016) IMDB data (break ties based on # of users, for example: 7.2 with 10,000 raters > 7.2 with 9,000 raters).

Draw a graph, where each dot is a film (i.e., 1,682 dots). The x-axis is the MovieLense ranking and the y-axis is today's IMDB ranking.

What is Pearon's r for the two lists (along w/ the p-value)? Assuming the two user bases are interchangable (which might not be a good assumption), what does this say about the attitudes about the films after nearly 20 years?

Problem 6

Repeat #6, but IMDB data from approximately July 31, 2005. What is the cumulative error (in days) from the desired target day of July 31, 2005? For example, if 1 memento is from July 1, 2005 and another memento is from July 31, 2006, then the cumulative error for the two mementos is 30 days + 365 days = 385 days.

Note: the URIs in the MovieLens data redirect, be sure to use the final values as URI-Rs for the archives:

$ curl -i -L --silent "http://us.imdb.com/M/title-exact?Top%20Gun%20(1986)"
HTTP/1.1 301 Moved Permanently
Date: Wed, 16 Mar 2016 18:37:06 GMT
Server: Server
Location: http://www.imdb.com/M/title-exact?Top%20Gun%20(1986)
Content-Length: 260
Content-Type: text/html; charset=iso-8859-1

HTTP/1.1 302 Found
Date: Wed, 16 Mar 2016 18:37:06 GMT
Server: HTTPDaemon
X-Frame-Options: SAMEORIGIN
Cache-Control: private
Location: http://www.imdb.com/title/tt0092099/
Content-Type: text/plain
Set-Cookie: uu=BCYuNIAbuc9FDeWcqVNAaaXLjXbagPPhyTQbhxr8CTOkHFcqkeyRbKqvk_m6buuHjm HkufNf5z5S4WGfKlG6BPOhzgA-jcsRZ5Q7GW2MJP0wNI9AZMnd245Mw_xI6spRuK_VF2lSxUGPIRXy4d-NY-YwZkqTEZ8uTOXchLSvqBpgsDI;expires=Thu, 30 Dec 2037 00:00:00 GMT;path=/;domain=.imdb.com
Vary: Accept-Encoding,User-Agent
P3P: policyref="http://i.imdb.com/images/p3p.xml",CP="CAO DSP LAW CUR ADM IVAo IVDo CONo OTPo OUR DELi PUBi OTRi BUS PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA HEA PRE LOC GOV OTC"
Content-Length: 0

HTTP/1.1 200 OK
Date: Wed, 16 Mar 2016 18:37:06 GMT
Server: Server
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: frame-ancestors 'self' imdb.com *.imdb.com *.media-imdb.com withoutabox.com *.withoutabox.com amazon.com *.amazon.com amazon.co.uk *.amazon.co.uk amazon.de *.amazon.de translate.google.com images.google.com www.google.com www.google.co.uk search.aol.com bing.com www.bing.com
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
Vary: Accept-Encoding,User-Agent
[deletia...]

References

[1] Segarn, Toby. Programming Collective Intelligence. *Building Smart Web 2.0 Application*. (pp 9). Sebastopol, CA: O'Reilly Media.