CSE6242 / CX4242: Data and Visual Analytics | Georgia Tech | Spring 2016

Homework 1: Analyzing The Movie DB dataset; SQLite; D3 Warmup; Gephi; OpenRefine

Due: Friday, February 5, 2016, 11:55PM EST

Prepared by Gopi Krishnan Nambiar, Nilaksh Das, Pradeep Vairamani, Ajitesh Jain, Vishakha Singh, Polo Chau

Submission Instructions:

It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or you may lose points.

- Submit a single zipped file, called "HW1-{YOUR_LAST_NAME}-{YOUR_FIRST_NAME}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: 'HW1-Doe-John.zip' if your name is John Doe. Only .zip is allowed (no .rar, etc.)
- You may collaborate with other students on this assignment, but you must write your own code and give the explanations in your own words, and also mention the collaborators' names on T-Square's submission page. All GT students must observe the honor code. **Suspected plagiarism and academic misconduct will be reported and directly handled** by the Office of Student Integrity (OSI). Here are some examples similar to Prof. Jacob Eisenstein's NLP course page (grading policy):
  - **OK:** discuss concepts (e.g., how cross-validation works) and strategies (e.g., use hashmap instead of array)
  - **Not OK:** several students work on one master copy together (e.g., by dividing it up), sharing solutions, or using solution from previous years or from the web.
- If you use any "*slip days*", you must write down the number of days used in the T-square submission page. For example, "Slip days used: 1". Each slip day equals 24 hours. E.g., if a submission is late for 30 hours, that counts as 2 slip days.
- At the end of this assignment, we have specified a folder structure about how to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- In your final zip file, please do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).
- After all slip days are used up, 5% deduction for every 24 hours of delay. (e.g., 5 points for a 100-point homework)
- We will not consider late submission of any missing parts of an homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check.

Part 1: Collecting & visualizing The Movie DB (TMDb) data

1. **[25 pt]** Use 'The Movie DB' API to: (1) download data about movies and (2) for each movie, download its 5 related movies.

a. How to use TheMovieDB API:
   - Create a TMDb account - https://www.themoviedb.org/account/signup . More detailed instructions can be found here.
   - After you have signed up, you need to request for an API key. Log into your account page on TMDb and generate a new key from within the "API" section found on the left sidebar.
     - While requesting for an API key(select "Developer"), you may enter:
       Type of Use - Education
       Application Name - "Homework 1"
       Application URL - http://poloclub.gatech.edu/cse6242/2016spring/
       Summary - "For the course CSE 6242"

     Once your request for the API key is approved, you can view the API key by clicking the "Details" tab under the "API" section.
   - Read the documentation (http://docs.themoviedb.apiary.io/#) to learn how to use the API.

     Note:
     - The API allows you to make 40 requests every 10 seconds. Set appropriate timeout intervals in the code while making requests.
     - The API endpoint may return different results for the same request.

b. [10 pt] Search for movies in the "**Science Fiction**" genre released in year 2000 or later. Retrieve the first 300 movies.
   - The documentation for movie search based on genre:
     http://docs.themoviedb.apiary.io/#reference/genres/genreidmovies
     http://docs.themoviedb.apiary.io/#reference/genres/genremovielist

   - Save the results in **`movie_ID_name.txt`**.
     Each line in the file should describe one movie, in the format:
     <movie-ID, movie-name>
     Column headers are not required.

c. [15 pt] For each of the 300 movies, use the API to find its first 5 similar movies (some movies might have fewer than 5 similar movies).
   - The documentation for obtaining similar movies:
     http://docs.themoviedb.apiary.io/#reference/movies/movieidsimilar

   - Save the results in **`movie_ID_sim_movie_ID.txt`**.
     Each line in the file should describe one pair of similar movies, in the format:
     <movie-ID, similar-movie-ID>
     Column headers are not required.

Note: You should remove all duplicate pairs. That is, if both the pairs `(A, B)` and `(B, A)` are present, only keep `(A, B)` where `A < B`. For example, if movie A has three similar movies X, Y and Z; and movie X has two similar movies A and B, then there should only be four lines in the file.

```
A, X
A, Y
A, Z
X, B
```

Deliverables:

Create a directory called **Q1** to store all the files listed below.

- **Code:** Write your own **program/script** to generate the two data files mentioned in steps b and c. You may use any language you like (e.g., shell script, python, Java). **Note:** Your submitted code should run as is (no extra installation or configuration should be required).

- Create a **README.txt** file to: (1) state the platform you have tested your code on (e.g., Windows 10, Mac OS X 10.11); (2) describe the instructions on how to run your code; (3) mention the API endpoints and query parameters used in step b and c.

- **Two data files**: **`movie_ID_name.txt`** and **`movie_ID_sim_movie_ID.txt`** produced in step b and c respectively.

2. **[20 pt]** Analyze the TMDb data in the files below, as an undirected graph (network). Download the files at:
   http://poloclub.gatech.edu/cse6242/2016spring/hw1/movie_ID_name_gephi.csv
   http://poloclub.gatech.edu/cse6242/2016spring/hw1/movie_ID_sim_movie_ID_gephi.csv

   Each movie in **`movie_ID_name_gephi.csv`** is represented as a node in the graph and each movie pair in **`movie_ID_sim_movie_ID_gephi.csv`** represents an undirected edge. You will visualize the graph of *similar movies* using Gephi (available for free download at http://gephi.org). *Note: Make sure your system fulfils all requirements for running Gephi.*

   Import all the edges in **`movie_ID_sim_movie_ID_gephi.csv`** by checking the "create missing nodes" option, since many of the IDs in **`movie_ID_sim_movie_ID_gephi.csv`** may not be present in **`movie_ID_name_gephi.csv`**. Note that in some cases, the data would need to be imported using the data lab rather than File -> Open

a. Gephi quick-start guide: https://gephi.org/users/quick-start/

b. [10 pt] Visualize the graph and submit a snapshot of a visually meaningful view of this graph. Here are some general guidelines for a visually meaningful graph:
   - Keep the edge crossing to minimum.
   - Keep the edge length roughly the same. Try and avoid very long edges.

- Keep the edges as straight lines without any bends.
- Keep the graph compact and symmetric if possible.
- All the nodes and edges should be properly visible.
- Make it easy to visualize which nodes any given node is connected to and also count its degree.
- Make it easy for any other tasks user might want to perform like finding the shortest path from one node to another or perhaps clusters of nodes with higher inter –connectedness, etc.

Experiment with Gephi's features, such as graph layouts, changing node size and color, edge thickness, etc. The objective of this task is to familiarize yourself with Gephi and hence is a fairly open ended task.

c. [6 pt] Using Gephi's built-in functions, compute and report the following metrics for your graph:
   - Average node degree
   - Diameter of the graph
   - Average path length

   Briefly explain the intuitive meaning of each metric in your own words.
   (These metrics will be discussed in the upcoming lectures on graphs.)

d. [4 pt] Run Gephi's in-built PageRank algorithm on your graph
   - Submit an image showing the distribution of the PageRank score.
     (The "distribution" is generated by Gephi's built-in PageRank algorithm where the horizontal axis is the PageRank score and the vertical axis is the number of nodes with a specific PageRank score)
   - List the top 5 movies (movie title and id) with the highest PageRank score

Deliverables:

Create a directory called **Q2** and place all the files listed below into it.

- **Result for part b:** An image file named "`movie_graph.png`" containing the snapshot of the graph of the data you obtained and a text file named "`graph_explanation.txt`" describing your choice of visualization, using fewer than 50 words.

- **Result for part c:** A text file named "`movie_graph_metrics.txt`" containing the three metrics and your intuitive explanation for each of them, using fewer than 150 words.

- **Result for part d:** An image file named "`movie_pagerank_distribution.png`" containing the distribution of the PageRank scores and append the movies with the 5 highest PageRank scores in the file "`movie_graph_metrics.txt`" from part c.

Part 2: Using SQLite

3. **[35 pt]** Elementary database manipulation with SQLite (http://www.sqlite.org/):
   a. [2 pt] *Import data:* Create an SQLite database called **rt.db**.
   Import the movie data from
   http://poloclub.gatech.edu/cse6242/2016spring/hw1/movie-name-score.txt
   into a new table (in rt.db) called **movies** with the schema:
   `movies(id integer, name text, score integer)`
   Import the movie cast data at
   http://poloclub.gatech.edu/cse6242/2016spring/hw1/movie-cast.txt
   into a new table (in rt.db) called **moviecast** with the schema:
   `moviecast(movie_id integer, cast_id integer, cast_name text)`
   Provide the SQL code (and SQLite commands used).

   Hint: you can use SQLite's built-in feature to import CSV (comma-separated values) files: https://www.sqlite.org/cli.html#csv

   b. [2 pt] *Build indexes*: Create two indexes that will also speed up consequent join operations:
   Index named movies_primary_index in movies table for `id` attribute
   Index named movies_secondary_index in moviecast table for `movie_id` attribute

   c. [2 pt] *Find average movie score:* Calculate the average movie score over all movies that have scores >= 80.

   Output format:
   `avg_score`

   d. [4 pt] *Finding poor films:* Find the 8 worst (lowest scores) movies which have scores > 50. Sort by score from lowest to highest, then name in alphabetical order.

   Output format:
   `id, name, score`

   e. [4 pt] *Finding laid back actors:* List 10 cast members (alphabetically by cast_name) with exactly 3 movie appearances.

   Output format:
   `cast_id, cast_name, movie_count`

   f. [6 pt] *Getting aggregate movie scores:* Find the top 10 (distinct) cast members who have the highest average movie scores. Sort by score (from high to low). In case of a tie in the score, sort the results based on name of cast members in alphabetical order. Filter out movies with score < 50. Exclude cast members who have appeared in fewer than 3 movies.

Output format:

```
cast_id, cast_name, avg_score
```

g. [7 pt] *Creating views:* Create a view (virtual table) called 'good_collaboration' that lists pairs of stars who appeared in movies. Each row in the view describe one pair of stars who have appeared in at least three movies together AND those movies have average scores >=75.

The view should have the format:

```
good_collaboration(cast_member_id1,cast_member_id2,avg_
                    movie_score, movie_count)
```

Exclude self pairs:

```
(cast_member_id1 == cast_member_id2)
```

Keep symmetrical or mirror pairs. For example, keep both (A, B) and (B, A).

Full points will only be awarded for queries that use joins.

Hint: Remember that creating a view will produce no output, so you should test your view with a few simple select statements during development.

h. [3 pt] *Finding the best collaborators:* Get the 5 cast members with highest average good_collaboration score from the view made in part g.

Output format:

```
(cast_id,cast_name,avg_good_collab_score)
```

i. [4 pt] Full Text Search (FTS) https://www.sqlite.org/fts3.html
Import the movie overview data from
http://poloclub.gatech.edu/cse6242/2016spring/hw1/movie-overview.txt
into a new FTS table (in rt.db) called **movie_overview** with the schema:

```
movie_overview(id integer, name text, year integer, overview
text, popularity decimal)
```

1. Find the count of all the movies which have the word "good" or "bad" in the overview field.
2. List the ids of the movies that contains the terms "life" and "about" in the overview field with not more than 5 intervening terms in between.

j. [1 pt] Explain your understanding of FTS performance in comparison with a SQL 'like' query and why one may perform better than the other. Please restrict your response to 50 words. Please save your responses in a file titled "**fts.txt"**

Deliverables:

Create a directory named **Q3** and place all the files listed below into it.

- **Code:**
  - A text file named "`Q3.SQL.txt`" containing all the SQL commands and queries you have used to answer questions a-j in the appropriate sequence. We will test its correctness in the following way:
    `$ sqlite3 rt.db <Q3.SQL.txt`
    Assume that the data files are present in the current directory.

  - **Important**: to make it easier for us to grade your code, after each question's query, append the following command to the txt file (which prints a blank line): `select '';` or `select null;`
    Here's an example txt file:

    ```
    Query for question a
    select '';
    Query for question b
    select '';
    Query for question c...
    ```

- **Answers:** A text file named "`Q3.OUT.txt`" containing the answers of the questions a - j. This file should be created in the following manner:
  - `$ sqlite3 rt.db < Q3.SQL.txt > Q3.OUT.txt`

  We will compare your submitted text file to the text created in the above manner.

Part 3: D3 Warmup and Tutorial

4. **[10 pt]** Go through the D3 tutorial here.

   Please complete steps 01-09 (Complete through "09. The power of data()").

   This is an important tutorial which lays the groundwork for homework 2.

   Hint: We recommend using Mozilla Firefox or Google Chrome, since they have relatively robust built-in developer tools.

Deliverables:

- **Code:** an entire D3 project directory. When run in a browser, it will display
  - 25 bars (as in step 9) with different color for each block of 5 bars (Unlike in the tutorial where all bars are of the same color, the first 5 bars will be of one color, the next 5 bars will be of a different color and so on.)
  - Your name which can appear above or below the bar chart.

  The files and folder structure for Q4 should be as specified at the end of the assignment.

Part 4: OpenRefine

5. **[10 pt]** Basic usage of OpenRefine:

a. Download OpenRefine http://openrefine.org/download.html

b. Import Dataset:
Launch Open Refine. It opens in a browser (127.0.0.1:3333).
   - Download the dataset from here: http://poloclub.gatech.edu/cse6242/2016spring/hw1/menu.csv
   - Choose "Create Project" -> This Computer -> "menu.csv". Click "Next".
   - You will now see a preview of the dataset. Click "Create Project" in the upper right corner.

c. Clean/Refine the data:
Note: OpenRefine maintains a log of all changes. You can undo changes. See the "Undo/Redo" button on the upper left corner.

i. [3 pt] Clean the "Event" and "Venue" columns (Select these columns to be Text Facets, and cluster the data). Write down your observations in fewer than 75 words.

ii. [2 pt] Use the Google Refine Evaluation Language to represent the dates in column (“date”) in the format mm/dd/yyyy.

iii. [1 pt] List a column in the dataset that contains only nominal data, and another column that contains ordinal data.

iv. [1 pt] Create a new column “URL” which contains a link to the dishes of a particular menu.
Format of new column entries: http://api.menus.nypl.org/dishes/139476 where 139476 is the “id”.

v. [3 pt] Experiment with Open Refine, and list a feature (apart from the ones used above) you could additionally use to clean/refine the data, and comment on how it would be beneficial in less than 50 words. Basic operations like editing a cell or deleting a row do not count.

Deliverables:
Create a directory named Q5 and place all the files listed below into it.
- Export the final table to a file named "Q5Menu.csv"
- Submit a list of changes “changes.json” made to file in json format. Use the “Extract Operation History” option under the Undo/Redo tab to create this file.
- A text file “Q5Observations.txt” with answers to parts c (i) , c (iii) and c (v).

Important Instructions on Folder structure

We will execute the following commands to validate the submission. Marks will be deducted if some files are missing in our query of the zip folder. For example, if Q1/movie_ID_name.txt is missing, 10pt could be deducted.

unzip -l HW1-LastName-FirstName.zip | grep -c txt should give 11 files.
Similarly for html, json, csv, js files as well.

The directory structure should look like:

```
HW1-LastName-FirstName/
    |--- Q1/
        |----  Your Script (script.py or script.js, etc.)
        |----  movie_ID_name.txt
        |----  movie_ID_sim_movie_ID.txt
        |----  README.txt
    |--- Q2/
        |----  graph_explanation.txt
        |----  movie_pagerank_distribution.png
        |----  movie_graph.png
        |----  movie_graph_metrics.txt
    |--- Q3/
        |----  fts.txt
        |----  movie-cast.txt
        |----  movie-name-score.txt
        |----  Q3.OUT.txt
        |----  Q3.SQL.txt
    |--- Q4/
        |----   index.html
        |----   d3/
                |----    d3.v3.min.js
    |--- Q5/
        |----  changes.json
        |----  Q5Menu.csv
        |----  Q5Observations.txt
```