CSE6242 / CX4242: Data and Visual Analytics | Georgia Tech | Spring 2013

Homework 1:
Analyzing Rotten Tomatoes Data; SQLite; D3 Warmup
Due: Monday, January 27, 2014, 11:59PM EST

Prepared by Robert Pienta, Tran Quoc Long, Polo Chau

Submission details: Submit each of the deliverables as a separate attachment on the t-square submission site with the *specified file name*. In case you have collaborated with other students on this assignment, please specify the name(s) of your collaborator(s) in the text box present on the t-square submission site. **Don't wait until the night before!!!**

Part 1: Collecting and visualizing Rotten Tomatoes data

1. **[30 pt]** Use the Rotten Tomatoes API to download data about movies and, for each movie, its 5 related movies.
    a. Information regarding the RottenTomatoes API:
        - Register at http://developer.rottentomatoes.com/page to access the API
            - Put the course name "CSE6242" as the application name and course website (http://poloclub.gatech.edu/cse6242/) as the application website for registering the application (this seems required). However, you may choose another name or website.
        - After you have registered and got an API key, look at the documentation (http://developer.rottentomatoes.com/docs) to find out how to use the API.
    b. [4 pt] Search for movies with the keyword "**star**" and retrieve the first 300 movies.
        - The documentation for keyword search: http://developer.rottentomatoes.com/docs/read/json/v10/Movies_Search
    c. [8 pt] For each of the 300 movies, use the API to find its 5 most similar movies.
        - The documentation for obtaining similar movies: http://developer.rottentomatoes.com/docs/read/json/v10/Movie_Similar
    d. Create the following two data files, from the obtained data
        - [8 pt] **`movie_ID_name.txt`**: each line describes one of the 300 movies, in the format: <movie-ID, movie-name>
        - [10 pt] **`movie_ID_sim_movie_ID.txt`**: each line describe one pair of similar movies obtained in step c, in the format: <movie-ID, similar-movie-ID>. Note that for each movie, there should be 5 or fewer similar movies. Remove all duplicate pairs, that is, if both the pairs `(A, B)` and `(B, A)` are present, only keep one of them (you can choose either one). 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:

- **Code:** you may use (1) Google Refine or (2) write your own program/script to generate the two data files that step d asks for; you can use any languages you like (e.g., shell script, python, Java);
  For either case, you must write a README file that include: (1) which approach you are using, (2) in case you write your own program/script, which platform you tested it on (e.g., Linux, Windows 8, etc.), and (3) instructions on how to run your code/script. Submit a **single zipped file**, called "`HW1.Q1.CODE.<extension>`", that contains your source code/script (unless you use Google Refine), and the **README** files.
- **Two data files**: "`movie_ID_name.txt`" and "`movie_ID_sim_movie_ID.txt`" produced in step d

2. **[20 pt]** We are going to view the data from question 1 as an undirected graph (network): each movie in **`movie_ID_name.txt`** as a node in the graph; and each movie pair in **`movie_ID_sim_movie_ID.txt`** as an undirected edge. You will visualize the graph of *similar movies* (based on the top 300 movies with the keyword "**star**") on Rotten Tomatoes using Gephi (available for free download at http://gephi.org). (*Clarification:* Please import all the edges in **`movie_ID_sim_movie_ID.txt.`** You have to check the "create missing nodes" option while importing the edges in Gephi since the many of the IDs in **`movie_ID_sim_movie_ID.txt`** will not be present in **`movie_ID_name.txt`**).
   a. Guide to getting started with Gephi: https://gephi.org/users/quick-start/
   b. [6 pt] Visualize the graph and submit a snapshot of a "visually meaningful" view of this graph. This is fairly open-ended and left to your preference. Experiment with Gephi's features, such as graph layouts, changing node size and color, edge thickness, etc.
   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
   d. [8 pt] Run Gephi's built-in PageRank algorithm on your graph
      - Submit an image showing the distribution of the PageRank score (*Clarification:* the "distribution" is the one 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 with the highest PageRank score

Deliverables:

- **Result for part b:** An image file named "`movie_graph.png`" containing the snapshot of the graph of the data you obtained from the rotten-tomatoes API.
- **Result for part c:** A text file named "`movie_graph_metrics.txt`" containing the three metrics
- **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. **[30 pt]** Elementary database manipulation with SQLite (http://www.sqlite.org/):
   a. [1 pt] *Import data:* Create an SQLite database called **rt.db**.
   Import the movie data at
   http://poloclub.gatech.edu/cse6242/2014spring/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/cse6242/2014spring/hw1/movie-cast.txt
   into a new table (in rt.db) called **cast** with the schema:
   `cast(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: http://www.sqlite.org/cvstrac/wiki?p=ImportingFiles

   b. [3 pt] *Find average movie score:* Calculate the average movie score over all movies that have scores >= 0.

   c. [3 pt] *Finding mediocre films:* Find the 5 worst (lowest scores) movies which have scores > 75. Sort by score, then name. Format of each output row:
   `id, name, score`

   d. [4 pt] *Finding laid back actors:* List 5 cast members (alphabetically by cast_name) with exactly 4 movie appearances. Format of each output row:
   `cast_id, cast_name, count_movies`

   e. [6 pt] *Getting aggregate movie scores:* Find the top 10 (distinct) cast members who have the highest average movie scores. List those cast members in alphabetical order. Filter out movies with score < 0. Format of each output row:

`cast_id, cast_name, average_score`

f. [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 two 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, count_movie)
```

Exclude self pairs:

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

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

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

g. [6 pt] *Finding the best collaborators:* Get the 15 cast members with highest average good_collaboration score from the view made in part f.
Output: `(cast_id,cast_name,average_good_collab_score)`

Deliverables:

- **Code:**
  - A text file named "`HW1.Q3.SQL.txt`" containing all the SQL commands and queries you have used to answer questions a-g in the appropriate sequence. We will test its correctness in the following way:
    `$ sqlite3 rt.db < HW1.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 '';`
    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 "`HW1.Q3.OUT.txt`" containing the answers of the questions a - g. This file should be created in the following way:
  - `$ sqlite3 rt.db < HW1.Q3.SQL.txt > HW1.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. Chad Stolper will be giving a lecture in class about D3 that will cover aspects used here and in homework 2.

   Please complete steps 01-08 (Complete through "08. Drawing Divs").

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

   Hint: We recommend using Firefox and Chrome; they have relatively robust built-in developer tools.

Deliverables:

- **Code:** an html file called "index.html" that, when run in a browser, will display
  - 5 bars with different dimensions (e.g., using divs); and
  - your name