CSE6242 / CX4242: Data and Visual Analytics | Georgia Tech | Spring 2015

Homework 1:
Analyzing Rotten Tomatoes Data; SQLite; D3 Warmup; OpenRefine

Due: Friday, January 30, 2015, 11:55PM EST

Prepared by Meera Kamath, Yichen Wang, Amir Afsharinejad, Chris Berlind, Polo Chau

Submission details: Submit a **single zipped file**, called "HW1-{YOUR_LAST_NAME}-{YOUR_FIRST_NAME}.zip", containing all the deliverables including source code/script, data files, and readme. Please read all instructions carefully about the deliverables for each question.

You may collaborate with other students on this assignment, but **each student must write up their own answers in their own words**, and must write down the collaborators' names on T-Square's submission page. If you are using "slip days", also write that down there.

Part 1: Collecting and visualizing Rotten Tomatoes data

1. **[25 pt]** Use the Rotten Tomatoes API to: (1) download data about movies and (2) for each movie, download its 5 related movies.
    a. How to use the RottenTomatoes API:
        - Register at http://developer.rottentomatoes.com/page to access the API.
            - While registering for an account, you may enter *CSE6242* for ***Application Name*** and *http://poloclub.gatech.edu/cse6242/* for ***Application URL***.
        - After you have registered and received an API key, look at the documentation (http://developer.rottentomatoes.com/docs) to learn how to use the API.

    Note:
        - The API allows you to make 5 calls per second. 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 with the keyword "**life**" and retrieve the first 300 movies.
        - The documentation for movie search based on keywords: http://developer.rottentomatoes.com/docs/read/json/v10/Movies_Search
        - Save the results in **`movie_ID_name.txt`**.

Each line in the file should describe one movie, in the format:
<movie-ID, movie-name>

c. [15 pt] For each of the 300 movies, use the API to find its 5 most similar movies (some movies might have fewer than 5 similar movies).
   - The documentation for obtaining similar movies: http://developer.rottentomatoes.com/docs/read/json/v10/Movie_Similar

   - Save the results in **`movie_ID_sim_movie_ID.txt`**.
     Each line in the file should describe one pair of similar movies obtained, in the format:
     <movie-ID, similar-movie-ID>.

     Note: You should remove all duplicate pairs. That is, if both the pairs `(A, B)` and `(B, A)` are present, keep only 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:

Create a directory named Q1 and place all the files listed below, including README.txt, into it.

- **Code:** Write your own **program/script** to generate the two data files mentioned in steps b and c. You may use any languages you like (e.g., shell script, python, Java). You must write a **README.txt** file that include: (1) which platform you have tested it on (e.g., Linux, Windows 8, etc.), and (2) instructions on how to run your code/script.
  Note: Using external libraries is allowed, but it should be included in the submission directory. The submitted code should run as-is. (No extra installation or configuration should be required.)

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

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 “**life**”) on Rotten Tomatoes using Gephi (available for free download at http://gephi.org).

Import all the edges in **`movie_ID_sim_movie_ID.txt`** by checking the “create missing nodes” option, since many of the IDs in **`movie_ID_sim_movie_ID.txt`** may not be present in **`movie_ID_name.txt`**.

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

b. [6 pt] Visualize the graph and submit a snapshot of a “visually meaningful” (and hopefully beautiful) 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

   Briefly explain the intuitive meaning of each metric in your own words. (We will discuss these metrics in upcoming lectures on graphs.)

d. [8 pt] Run Gephi’s built-in 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 with the highest PageRank score

Deliverables:

Create a directory named 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 from the rotten-tomatoes API and a text file named “graph_explanation.txt” describing your choice of visualization, using fewer than 50 words (e.g. layout, color).

- **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 100 words total.

- **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. [2 pt] *Import data:* Create an SQLite database called **rt.db**.
      Import the movie data from
      http://poloclub.gatech.edu/cse6242/2015spring/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/cse6242/2015spring/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. [2 pt] *Build indexes*: Create two indexes over the table movies.
      The first index named movies_name_index is for the attribute name.
      The second index named movies_score_index is for the attribute score.

   c. [2 pt] *Find average movie score:* Calculate the average movie score over all movies that have scores >= 1.

   d. [4 pt] *Finding mediocre films:* Find the 5 worst (lowest scores) movies which have scores > 80. 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 5 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 < 1. Exclude cast members who have appeared in fewer than 3 movies.

Output format:
`cast_id, cast_name, average_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).

   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 necessary.

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,average_good_collab_score)`

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-h 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 '';`
    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 - h. This file should be created in the following way:
  - `$ 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. Chad Stolper will give a lecture on 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 entire D3 project directory. When run in a browser, it will display
  ○ 5 bars (as in step 8) with different bar color (not the green one used in the tutorial) and your name.
  ○ The directory structure should look like:

```
Q4/
   |----   index.html
   |----   d3/
              |----   d3.v3.min.js
```

Part 4: OpenRefine

5. **[15 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/cse6242/2015spring/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. [4 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. [3 pt] The values in the "Date" column do not follow a consistent format. Some values are of the form yyyy-mm-dd, others mm/dd/yy. Use the Google Refine Evaluation Language to format the cell values to represent dates in the format mm/dd/yyyy, and ensure it is in the right format.

iii. [2 pt] List a column in the dataset that contains only nominal data, and another column that contains ordinal data.

iv. [2 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".
Note: This link will not work without the API key. You are welcome to get an API key to test the link, but this is not required for the assignment.

v. [4 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. 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 made to file in json format.
- A text file "Q5Observations.txt" with answers to parts c (i) , c (iii) and c (v).