CSE6242 / CX4242: Data and Visual Analytics | Georgia Tech | Fall 2014

Homework 1:
Collecting and Visualizing Data; SQLite; D3 Warmup

Due: Monday, September 8, 2014, 11:59PM EDT

Prepared by Brian Minsuk Kahng, Seungyeon Kim, Alan Zhang, Drew Wei, Polo Chau

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

If you have collaborated with other students, write down their name(s) in the text box on the t-square submission site. **Each student must write his/her own answers.**

Part 1: Collecting and Visualizing Last.fm Data

1. **[30 pt]** Use the Last.fm API to download data about music tracks, and 100 similar tracks for each track.
   a. Get access and become familiar with the Last.fm API:
      - Register at http://www.last.fm/api to get an API account.
      - Read the doc for "track.getSimilar" API.
   b. [4 pt] Get 100 similar tracks for **Cher**'s **Believe**.
      - The documentation for getting similar tracks: http://www.last.fm/api/show/track.getSimilar
      - Discard any track that does not have "mbid". This may result in less than 100 tracks. You do not need to crawl additional tracks to make it 100.
   c. [8 pt] For each of the 100 tracks, use the API to find its 10 most similar tracks. You should set some wait time between any two consecutive requests. Discard any track that does not have "mbid". Skip some songs when the API returns "No Track Found" with a mbid.
   d. Create the following two data files, from the obtained data
      - [8 pt] **`tracks.csv`**: each line describes one of the 100 tracks, in the following format. (The first line should be the header.) **Use UTF-8 as the file's character encoding**, since some track names and artists contain unicode characters.

```
id,track_name,artist
03bc1da8-3b25-4060-b843-1fbf60c046a8,"Strong Enough","Cher"
a88735e6-b35c-4379-8ef7-bbd2b793ccf4,"All or Nothing","Cher"
```

      - [10 pt] **`track_id_sim_track_id.csv`**: each line describes one pair of similar tracks obtained in step **c**, in this format:

```
source,target
03bc1da8-3b25-4060-b843-1fbf60c046a8,a88735e6-b35c-4379-8ef7
-bbd2b793ccf4
```

Note that for each track, there should be 10 or fewer similar tracks. Discard all tracks that do not have a "mbid" and 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 track #1 has three similar tracks #3, #4 and #5; and track #3 has two similar tracks #1 and #2, then there should only be four lines in the file:

```
#1,#3
#1,#4
#1,#5
#2,#3
```

Deliverables:

- **Code:** You need to write your own program/script to generate the two data files which step **d** asks for; you can use any languages you like (e.g., python, shell script, Java); You must write a README.txt file which includes: (1) which platform you tested it on (e.g., Linux, Windows 8, etc.) and (2) instructions on how to run your code/script. Create a directory named Q1 and place all the files, including README.txt, into it.
- **Two data files**: result files: "`tracks.csv`" and "`track_id_sim_track_id.csv`" produced in step **d**

2. **[20 pt]** We are going to view the data from question 1 as an undirected graph (network): each track in **`tracks.csv`** as a node in the graph; and each track pair in **`track_id_sim_track_id.csv`** as an undirected edge. You will visualize the graph of *similar tracks* obtained from question 1 using Gephi (available at http://gephi.org). (*Clarification:* Please import all the edges in **`track_id_sim_track_id.csv.`** You have to check the "create missing nodes" option while importing the edges in Gephi since many of the IDs in **`track_id_sim_track_id.csv`** will not be present in **`tracks.csv`**). *Gephi may not work properly with Java version 1.7 or higher. You may need to "downgrade" Java to 1.6 (For example, if you are a Mac user, you can go to http://support.apple.com/kb/DL1572 to install Java 1.6; then go to JAVA_HOME, most likely /Library/Java/JavaVirtualMachines and only keep the java 1.6 folder).*
    a. Guide to getting started with Gephi: https://gephi.org/users/quick-start/
    b. [10 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. In addition, filter out nodes whose degrees are very small.
    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. [4 pt] Run Gephi's built-in PageRank algorithm on your graph.
        - List the top 10 tracks with the highest PageRank score

Deliverables:

- **Result for b:** An image file named “`graph.png`” containing the snapshot of the graph of the data you obtained from the Last.fm API, and a text file named “`graph_explanation.txt`” describing your choice of visualization (e.g. layout, color, etc.) (less than 50 words)
- **Result for c:** A text file named “`graph_metrics.txt`” containing the three metrics
- **Result for d:** Append the tracks with the 10 highest PageRank scores to the file “`graph_metrics.txt`” from **c**.

Part 2: Using SQLite with Rotten Tomatoes Data

3. **[35 pt]** Elementary database manipulation with SQLite (http://www.sqlite.org/):
    a. [3 pt] *Import data:* Create an SQLite database called **Q3.db**.
    Import the movie data at
    http://poloclub.gatech.edu/cse6242/2014fall/hw1/Q3_movies.csv
    into a new table (in Q3.db) called **movies** with the schema:
    ```
    movies(id integer primary key, name text,
           year integer, genre text, score integer)
    ```
    Import the actors data at
    http://poloclub.gatech.edu/cse6242/2014fall/hw1/Q3_actors.csv
    into a new table (in Q3.db) called **actors** with the schema:
    ```
    actors(id integer primary key, name text)
    ```
    Import the movie cast data at
    http://poloclub.gatech.edu/cse6242/2014fall/hw1/Q3_cast.csv
    into a new table (in Q3.db) called **cast** with the schema:
    ```
    cast(movie_id integer, actor_id integer, character_name text,
        primary key (movie_id, actor_id))
    ```
    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. [3 pt] *Find average movie scores for each genre:* Calculate the average movie scores for each movie that have score > 0.
    Format of each output row:
    `genre, average_score`

d. [3 pt] *Finding recently released popular films:* Find the 10 best (highest scores) movies that was released from 2011 to 2014. Sort by score from high to low, then name (in alphabetical order).
   Format of each output row:
   `id, name, year, score`

e. [5 pt] *Finding diligent actors:* List all actors (alphabetically by name) with 10 or more movie appearance. Format of each output row:
   `actor_id, name, count_movies`

f. [6 pt] *Getting aggregate movie scores:* Find the top 10 actors who have the highest average movie scores. Filter out movies with score = 0. Exclude actors who appeared in less than 3 movies. Sort by score (from high to low). Format of each output row:
   `actor_id, name, average_score`

g. [8 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(actor_id1, actor_id2,
                   avg_movie_score, count_movies)
```

   Keep symmetrical or mirror pairs. For example, keep both (A, B) and (B, A). Make sure to exclude self pairs `(actor_id1 == actor_id2)`.

   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. [5 pt] *Finding the best collaborators:* Get the 5 actors with highest average good_collaboration score from the view made in **g**.
   Output: `(actor_id, name, average_good_collab_score)`

Deliverables:

- **Code:**
  - A text file named "`Q3.sql`" 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 Q3.db < Q3.sql`
    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 sql 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 - g. This file should be created in the following way:
  - `$ sqlite3 Q3.db < Q3.sql > 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. **[15 pt]** Go through the D3 tutorial at http://alignedleft.com/tutorials/d3/. 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 or 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
```