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

Homework 1: Analyzing Twitter dataset; SQLite; D3 Warmup; Gephi; OpenRefine

Due: Friday, September 16, 2016, 11:55PM EST

Prepared by Nilaksh Das, Pradeep Vairamani, Vishakha Singh, Yanwei Zhang, Bhanu Verma, Meghna Natraj, Polo Chau

Submission Instructions and Important Notes:

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 to 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.
- We will use auto-grading scripts to grade some of your deliverables (there are hundreds of students), so it is extremely important that you strictly follow our requirements. Marks may be deducted if our grading scripts cannot execute on your deliverables.
- 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 a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check.

Part 1: Collecting and visualizing Twitter data [45 pt]

1. **[30 pt]** You will use the Twitter REST API to retrieve (1) *followers*, (2) *followers of followers*, (3) *friends* and (4) *friends of friends* of a user on Twitter (a Twitter *friend* is someone you follow and a Twitter *follower* is someone who follows you).

a. The Twitter REST API allows developers to retrieve data from Twitter. It uses the OAuth mechanism to authenticate developers who request access to data. Here's how you can set up your own developer account to get started:

- Create a Twitter account, if you don't already have one.
- Now you need to get API keys and access tokens that uniquely authenticate you. Sign into Twitter Apps with your Twitter account credentials. Click '*Create New App*'. While requesting access keys and tokens, enter:

Name	dva_hw1_<your-student-id> (eg: dva_hw1_jdoe3)
Description	"For CSE 6242 @ GaTech"
Website	http://poloclub.gatech.edu/cse6242/2016fall/
Callback URL	field should be left empty as we won't be needing it

Check the developer agreement checkbox and click on *'Create your Twitter application'*. Once your request is approved, you can click *'Keys and Access Tokens'* to view your *'API key'* and *'API secret'*. You will also need to generate your access token by clicking the *'Create my access token'* button. After this step, you are ready to make authenticated API calls to fetch data.

Store the credentials that you just created in a **keys.json** file which you will use later. The file should have the following contents:

```
{
        "api_key": "your api key here",
        "api_secret": "your api secret here",
        "token": "your access token here",
        "token_secret": "your access token secret here"
}
```

Important notes and hints:

- Twitter limits how fast you can make API calls. For example, the limit while making GET calls for friends is 15 requests per 15 minutes.
- Refer to the rate limits chart for different API calls.
- Set appropriate timeout intervals in the code while making requests.
- An API endpoint may return different results for the same request.

You will use **Python 3.0+** and the tweepy library to accomplish the following tasks. We have created a boilerplate script to help you get started and to reduce the amount of code you may need to write. You will modify parts of this script for the tasks below. Download

the boilerplate code from this link.
If you are new to Python, here are few useful links to help you get started:

- http://www.tutorialspoint.com/python3/
- https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3/File_IO

b. [15 pt] Search for followers of the Twitter screen name "*PoloChau*". Use the API to retrieve the first 10 followers. Further, for each of them, use the API to find their 10 followers.

- Read the documentation for getting followers of a Twitter user.
- You code will write the results to **followers.csv**.
  Each line in the file should describe one relationship in the format:
  `follower-screen-name, user-name`
- Grading distribution is given in the boilerplate code.

  Note: *follower-screen-name* represents the source and *user-name* represents the target for an edge in a directed graph. You will be adding these column headers to the CSV file in a later question.

c. [15 pt] Search for friends of the Twitter screen name "*PoloChau*". Use the API to retrieve the first 10 friends. Further, for each of the 10 friends, use the API to find their 10 friends.

- Read the documentation for getting friends of a Twitter user.
- You code will write the results to **friends.csv.**
  Each line in the file should describe one pair of relationship in the format:
  `user-name, friend-screen-name`
- Grading distribution is given in the boilerplate code.

  Note: *user-name* represents the source and *friend-screen-name* represents the target for an edge in a directed graph. You will be adding these column headers to the CSV file in a later question.

If a user has fewer than 10 followers or friends, the API will return as many as it can find. Your code should be flexible to work with whatever data the API endpoint returns.

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

- **script.py:** The boilerplate code modified by you. The submitted code should run as is. That is, no extra installation or configuration should be required other than the specified libraries. Also specify the python version in the code.

- **followers.csv** and **friends.csv** produced in step b and c respectively. Please note that these files will be modified in task 2b shortly.

  Note: Do **NOT** submit your API credentials (**keys.json**). They should not be shared. We will use our own keys and tokens to grade your work.

2. **[15 pt]** Visualize the network of friends and followers obtained previously using Gephi,

which you can download for free.
Note: Make sure your system fulfils all requirements for running Gephi.

a. Go through the Gephi quick-start guide.

b. [2 pt] Insert `Source, Target` as the first line in both **followers.csv** and **friends.csv**. Each line in both files now represents a directed edge with the format `source, target`. Import all the edges contained in these files using Data Laboratory.
Note: Remember to check the "create missing nodes" option while importing since we don't have an explicit nodes file.

c. [8 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 edge crossing to a minimum, and avoid as much node overlap as possible.
   - Keep the graph compact and symmetric if possible.
   - Whenever possible, show node labels. If showing all node labels create too much visual complexity, try showing those for the "important" nodes.
   - Using colors, sizes, thicknesses, etc. to convey information.
   - Using nodes' spatial positions to convey information (e.g., "clusters" or groups).

   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.

d. [5 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.
   You will learn about these metrics in the "graphs" lectures.

Deliverables: Place all the files listed below in the **Q1** folder.

- **Result for part b: followers.csv** and **friends.csv** (with `Source, Target` as their first lines).

- **Result for part c:** An image file named "`graph.png`" (or "`graph.svg`") containing your visualization and a text file named "`graph_explanation.txt`" describing your design choices, using no more than 50 words.

- **Result for part d:** A text file named "`metrics.txt`" containing the three metrics and your intuitive explanation for each of them, using no more than 100 words.

Part 2: Using SQLite [35 pt]

The following questions help refresh your memory about SQL or get you started with SQLite, which is a lightweight, serverless embedded database that can easily handle up to multiple GBs of data. SQLite is great for building prototypes and sharing data (all data stored in a single cross-platform file).

a. [2 pt] *Import data:* Create an SQLite database called **rt.db**.

   Import the movie data from
   http://poloclub.gatech.edu/cse6242/2016fall/hw1/data/movies.txt
   into a new table (in rt.db) called **movies** with the schema:

```
movies(movie_id integer, name text, genre text)
```

   Import the movie rating data from
   http://poloclub.gatech.edu/cse6242/2016fall/hw1/data/ratings.txt
   into a new table (in rt.db) called **ratings** with the schema:

```
ratings(user_id integer, movie_id integer, rating real,
            timestamp integer)
```

   Provide the SQL code (and SQLite commands used).

   Data source: http://grouplens.org/datasets/movielens

   Note: You can use SQLite's built in feature to import data from files (https://www.sqlite.org/cli.html#section_3: *.separator STRING* and *.import FILE TABLE*)

b. [2 pt] *Build indexes*: Create two indexes that will speed up subsequent join operations:

   An index called *movies_primary_index* in the **movies** table for the `movie_id` attribute
   An index called *movies_secondary_index* in **ratings** table for the `movie_id` attribute

c. [2 pt] Find the total number of movies that are reviewed by at least 500 reviewers and with average ratings >= 3.5.

   Output format:

```
movie_count
```

d. [4 pt] *Finding most reviewed movies*: List all the movies with at least 2500 reviews. Sort the movies by the review count (high to low) then by their names (alphabetical order) for those who may have the same review counts.

   Output format:

```
movie_id, movie_name, review_count
```

e. [4 pt] *Finding best films:* Find the top 10 movies (highest average ratings). Sort the movies

by their average ratings (high to low) then by their names (alphabetical order).

Output format:
`movie_id, movie_name, avg_rating`

f. [5 pt] *Finding the best movies with the most reviews*: Find the top 8 movies with the highest average ratings that are rated by at least 1000 users. Sort the results by the movies' average rating (from high to low), then by the movies' names (alphabetical order), and then genres (alphabetical order).

Output format:
`movie_name, avg_rating, review_count, movie_genre`

g. [7 pt] *Creating views*: Create a view (virtual table) called *common_interests* from the data, such that: for each movie with exactly 10 reviews, show its reviewers in pairs, for all unique reviewer combinations. User IDs should be ranked in ascending order, and within a pair, the first user ID should be strictly smaller than the second ID. For example, movie M has 10 reviews, rated by reviewers 1,2,3,4,5,6,7,8,9,10. You would show "(1, 2, M)", "(1, 3, M)", …, "(1, 10, M)", "(2, 3, M)", ... , "(2, 10, M)", etc. This example has 45 such pairs.

The view should have the format:

```
common_interests(user_id1, user_id2, movie_name)
```

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

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

h. [2 pt] Calculate the total number of such pairs created from the view made in part *g*.

Output format:
`common_interest_count`

i. [4 pt] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation).

Import the movie overview data from 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. Count the number of movies whose *overview* fields contain the word "death" or "life".
2. List the ids of the movies that contain the terms "life" and "about" in their *overview* fields with no fewer than 6 intervening terms in between.

j. [3 pt] Explain your understanding of FTS performance in comparison with a SQL 'like' query and why FTS may perform better (hint: try SQLite's EXPLAIN command). Write down your explanation in fewer than 50 words in "**fts.txt"**.

Deliverables: Place all the files listed below in the **Q2** folder

- **Code:** A text file named "`Q2.SQL.txt`" containing all the SQL commands and queries you have used to answer questions a - i in the appropriate sequence. We will test its correctness in the following way:
  `$ sqlite3 rt.db < Q2.SQL.txt`
  Assume that the data files are present in the current directory.

  Note: We will use auto-grading scripts to grade your code (there are hundreds of students), so it is extremely important that your code strictly follow the requirements below. Marks may be deducted if our grading scripts cannot execute on your code.
  1. You should set up the separator for your output format as: `.separator ', '`
  2. After each question's query, append the following command to the txt file (which prints a blank line): `select '';` or `select nul l;`

  Here's an example txt file:

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

- **Answers:**
  1. A text file named "`Q2.OUT.txt`" containing the answers of the questions a - i. This file should be created in the following manner:
     `$ sqlite3 rt.db < Q2.SQL.txt > Q2.OUT.txt`
  2. A text file named "`fts.txt`" containing the answer to the question j.

  Note: We will compare your submitted text file to the text created in the above manner. Please follow the output format. We will strictly follow the requirements when grading.

Part 3: D3 Warmup and Tutorial [10 pt]

- Go through the D3 tutorial here.
- Complete steps 01-09 (Complete through "09. The power of data()").
- This is a simple and important tutorial which lays the groundwork for Homework 2.

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

Deliverables: Place all the files/folders listed below in the **Q3** folder

- A folder named **d3** containing file **d3.v3.min.js** (download)
- **index.html** : When run in a browser, it should display:
    - 25 bars (as in step 9) with different color for each block of 5 bars (the first 5 bars will be of one color, the next 5 bars will be of some other random color); where each bar should have a black solid border, width=30px and margin between two bars=2px.
    - Your name which can appear above or below the bar chart.

Note: No external libraries should be used. The index.html file can only refer to d3.v3.min.js within the d3 folder.

Part 4: OpenRefine [10 pt]

a. Watch the videos on the OpenRefine's homepage for an overview of its features. Download OpenRefine (latest release : 2.6 r.c2)

b. Import Dataset:
    - Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
    - Download the dataset
    - 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" column (Select the column to be a Text Facet, and cluster the data. *Note: You can choose different "methods" and "keying functions" while clustering)*

1. [2 pt] Observe the differences in the 2 methods (key collision and nearest neighbour) and cluster the data using the method that you think is better. State 2 reasons why you used this method. Describe your observations in fewer than 50 words.

2. [1 pt] Using the method "key collision", observe the differences while using the keying functions (a) fingerprint, and (b) "n-gram fingerprint" with n-gram size=2 . Describe a main difference you observed, in fewer than 30 words.

ii. [2 pt] Use the General Refine Evaluation Language (under Edit Cells → Transform) to represent the dates in column ("date") in a format such that "1900-04-15" is converted to "Sunday, April 15, 1900"

iii. [1 pt] List a column in the dataset that contains only nominal data, and another column that contains only ordinal data. (Refer to their definitions here)

iv. [1 pt] Create a new column called *URL* that contains links to the dishes of a particular menu. The values in that column should follow the following format (without the quotes) "`https://api.menu.nylp.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 fewer than 50 words. (*Basic operations like editing a cell or deleting a row do not count.)*

Deliverables: Place all the files listed below in the **Q4** folder

- **Q4Menu.csv** : Export the final table.
- **changes.json** : Submit a list of changes made to file in json format. Use the "*Extract Operation History*" option under the Undo/Redo tab to create this file.
- **Q4Observations.txt** : A text file with answers to parts c (i) and c (iii) and c (v)

Important Instructions on Folder structure

The directory structure must be:

```
HW1-LastName-FirstName/
    |--- Q1/
        |----  followers.csv
        |----  friends.csv
        |----  graph.png / graph.svg
        |----  graph_explanation.txt
        |----  metrics.txt
        |----  script.py
    |--- Q2/
        |----  fts.txt
        |----  Q2.OUT.txt
        |----  Q2.SQL.txt
    |--- Q3/
        |----   index.html
        |----   d3/
            |----    d3.v3.min.js
    |--- Q4/
        |----  changes.json
        |----  Q4Menu.csv
        |----  Q4Observations.txt
```