UCR-CS172 Final Project

Sabrina Wong
861195047

Part 1 - Crawler

Overview of system

Using Twitter Stream API, I created a stream that retrieves realtime tweets geo-tagged in the San Francisco, CA area. In the `twitter_streaming.py`, the program streams for tweets and inserts into `fetched_tweets.json`.

Retrieving 1GB is hard to store in GitHub. `twitter_streaming.py` is set to stream 1MB of tweets for the tester to create a small file. The console output of `twitter_streaming.py` is the file size. Average runtime 15-20 for 1MB.

Limitations (if any) of the system.

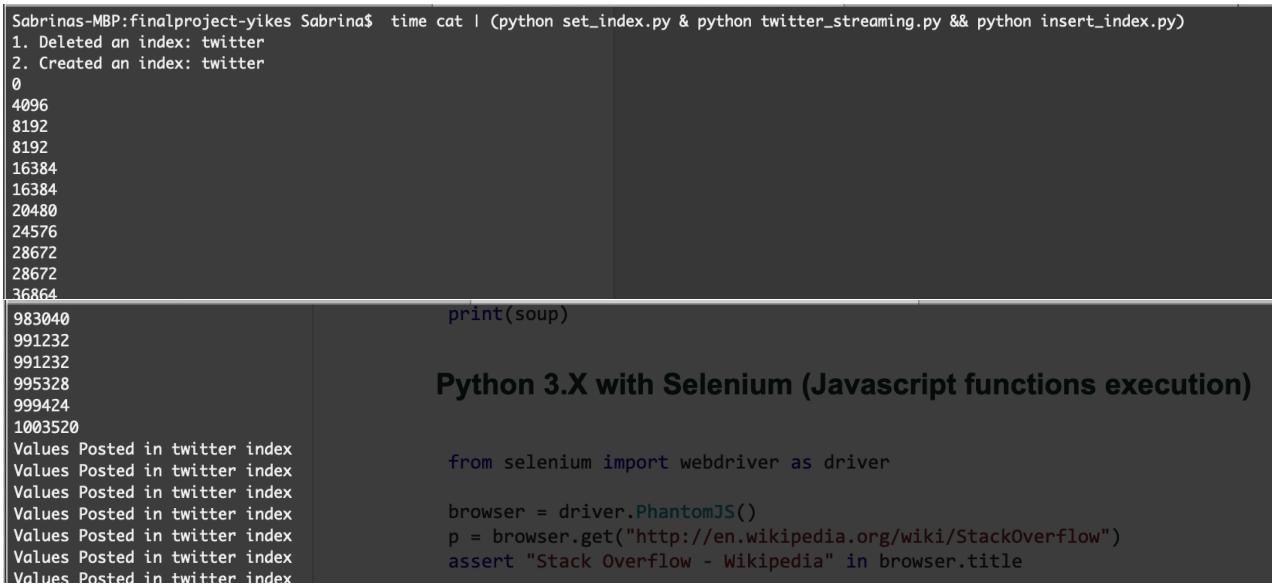
- `fetched_tweets.json` does not contain [urlTitles] of the tweets.
- does not allow user input on how many tweets to retrieve in crawler; instead it is a fixed size.

Instruction on how to deploy the crawler.

While ElasticSearch is running, to use the crawler and insert its retrieval into the index, run:

```
crawl_index.sh
```

Screenshots showing the system in action.



Part 2 - Indexer

Overview of system

The index to store the retrieved tweets is created through ElasticSearch. `set_index.py` is a file that initializes the setting of the index. `insert_index.py` reads through the `fetched_tweets.json`, analyzes each json in the file, and inserts it into ElasticSeach.

In the process of parsing, selected objects like [tweet id], [screen name], [location], [text], and [hashtags]. Text retrieval is based if the tweet is truncated because of the 140-character limit.

If a tweet contains any hashtags, it is included in the index stored as an array. If a tweet contains URLs, a function is called to parse the link and store the title of the tweet. Because tweets can contain URLs that link to images, titles are stored based on the elements stored in [entities: urls].

To maintain a sizeable index, the index should be deleted and re-created every time `twitter_streaming.py` has run.

Limitations of system

- The index only stores [tweet id], [screen name], [location], [text] from tweet, [timestamp] of tweet, [hashtags], and [urlTitles] of tweets rather than storing all the data from the json.

Instructions on how to deploy the indexer.

While ElasticSearch is running, to insert an already existing .json into the index, run:

```
json_index.sh
```

.json must be in data/ directory.

Part 3 - Extension

As an extension to the parts above, I created a web-based interface using Flask, a python web-development framework. This interface allows the user to see all the tweets retrieved from the index in the first page. Upon querying, the resulting page shows tweets relevant to the user's input and a score on the right side. Scoring, sorting, and ranking are all done with ElasticSeach. Scoring is the sum of all fields relevant to query.

Limitations of system

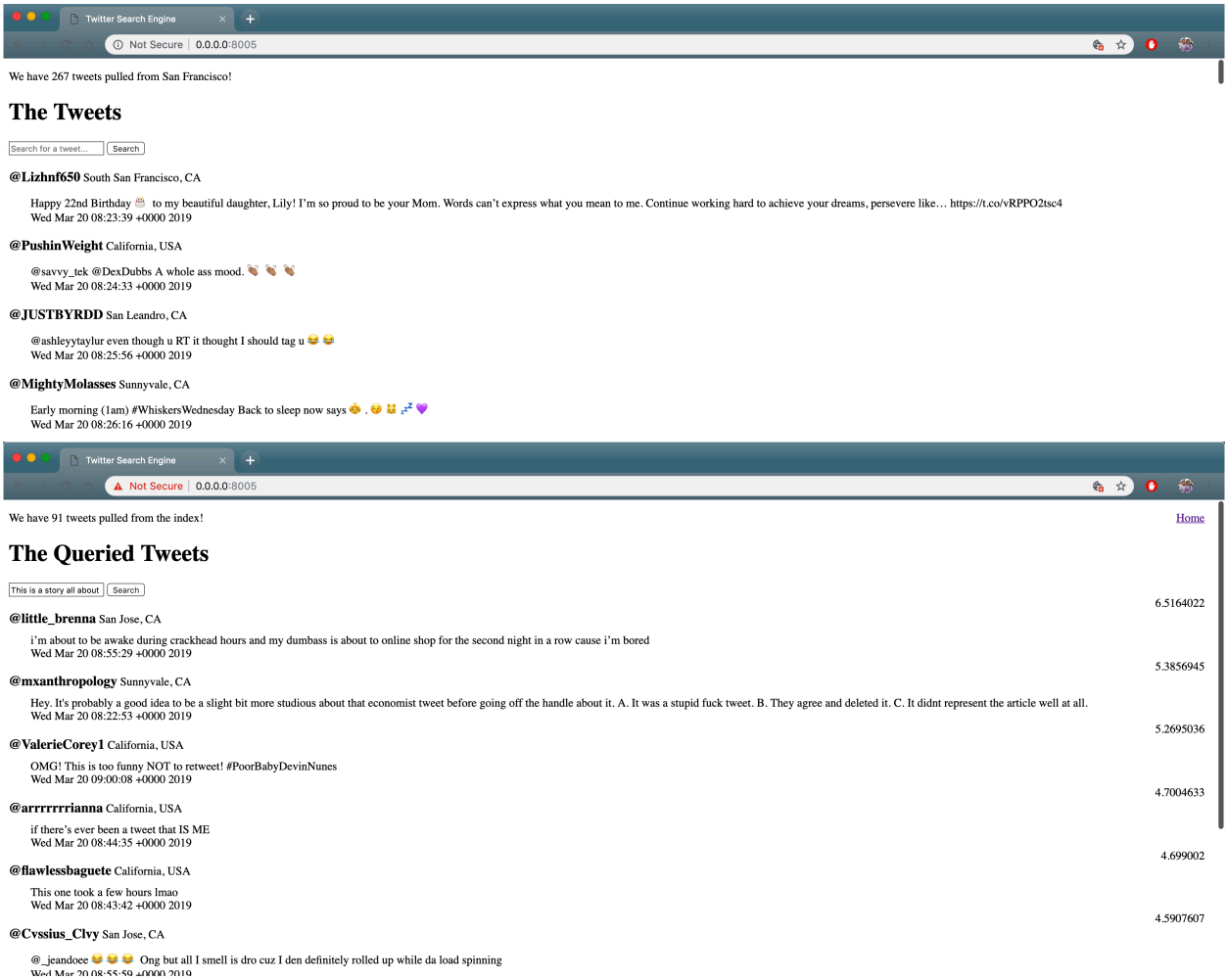
- This is a very, very basic UI.
- Displayed tweet attributes are [screen name], [location], [text], and [timestamp]. [score] is also displayed to the right, upon query.
- Searching helpers like "query" and boolean AND OR NOT are not implemented.
- Not category specific
- Some issues that arose are:
    - Querying emojis: this UI doesnt recognize emoji's
    - Doesn’t show the words queried before listing results

Instructions on how to deploy the system.

// command to start docker and run .sh , in that order

Open http://0.0.0.0:8005 in browser of choice.

Screenshots showing the system in action.



Part 4 - Other remarks

In the future, I would like to improve upon this system by:

- improving and expanding from the limitations stated in previous parts.
    - include a category/field option for user querying.
    - improve on UI (prettying up the front end, etc.).
    - allow user input on how many tweets to retrieve in crawler.