CS172 Part 2

Nathaniel Sakissian SID: 861190587

Lin Wang SID: 861074512

Clay Yantzer SID: 861068717

**Collaboration Details:**

Nathaniel Sarkissian:

Built index with Whoosh and developed search function in Python.

Built documentation.

Lin Wang:

Built documentation, cooperated in building index and interface.

Clay Yantzer:

Built user interface and integrated map features.

**Overview:**

Architecture:

Back end:

All indexing and searching was done in Whoosh with Python.

The city name to coordinate conversion was done with the Geocoder library.

Front end:

Build web-based interface in Flask library and HTML with python

The map integration was done with the flask-googlemaps library.

Index Structures:

Whoosh uses the Okapi BM25 scoring function by default, but also allows the option for a few others including tf-idf. We used the BM25 because it produced good results for our data. In order to also use the time of the tweets as well, we added a time factor to the BM25 score. The time factor is calculated by subtracting the tweet time(in milliseconds since epoch) from the current time(in milliseconds since epoch), and reciprocating it.

timeFactor = 1/(currentTime - tweetTime)

The time factor is also weighted in order to balance its relevance compared to the BM25 score.

Search Algorithm:

BM25 score:

Given a query Q, with keywords q1 … qn, the BM25 score is

is the ith term frequency in document D. |D| is the length of document D. avgdl is the average document length in the text collection from the drawn documents. K1 and b are free parameters.

N is the total number of documents in collection, n(q) is the number of documents containing q.

Combination of BM25 and Time:

This combine the relevance score and time elapsed

score(time) = 1/(currentTime - tweetTime). The most recent posts will have the largest values.

Overall score = Score(BM25) + Score(time)

**Limitations of system:**

One major limitation of our system is that it pays no attention to duplicates. Furthermore, the duplicates produced by retweets are also not handled. For this reason, the search may return the same tweet more than once.

Considering of running time, we set limitations to read first 1000 tweets for each text file to form index. The limits can be increased or deleted.

This is not dynamic web interface.

Most tweets are not geo-located, so the majority of searches will not display any markers on the map.

**Instructions on how to deploy the system:**

1. To deploy the whole system, need to pip install whoosh, flask, geocoder, flask-googlemaps

2. Run the indexSearch.py with cmd: python indexSearch.py

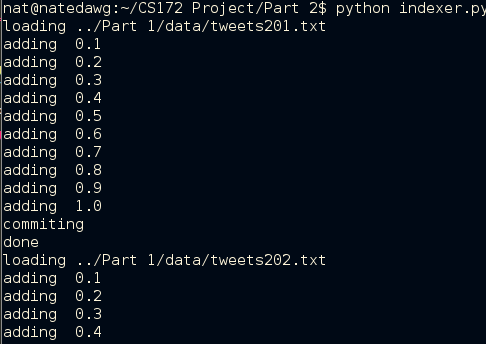
This will create index for all the data text files with tweets.

3. Run the Web interface: python app.py

4. Put search query into the search box.

**Screenshots:**

This is what the indexer looks like while adding the documents.



The user enters a search term and hits send.



The results are listed, along with a map. The score of each tweet is displayed in the far left column.

