Health Monitor Analytics

GROUP 1

Demo 1

Technical Documentation

D. Takacs

G. Bati

J. Abdulbaqi

I. Paraskevakos

K. Dong

Table of Contents

1. Introduction

The objective of this technical documentation is to describe, from a programmer's standpoint, how different components of the code function individually and together to form a unified unit. It is assumed that a programmer has a good experience on Python how to understand its code as well as how to compile it.

2. Interactive Map

This part using the code of the following projects and services:

- Twitter Streaming API (https://dev.twitter.com/docs/streaming-apis)
- Google Maps API (https://developers.google.com/maps/)
- The famous JavaScript library - JQuery (http://jquery.com)
- The jquery.eventsource for eventsource API using for opening an HTTP connection for receiving push notifications from a server in the form of DOM events (https://github.com/rwaldron/jquery.eventsource).
- JavaScript Heatmap Library (http://www.patrick-wied.at/static/heatmapjs/)
- twitter-realtime-heatmap project (https://github.com/comsysto/twitter-realtime-heatmap)

In addition to the previous code, this feature uses the following server and technique:

- Flask (http://flask.pocoo.org/)
- Redis (http://redis.io/)
- Tweepy (http://tweepy.github.com/)

The feature code written by using python, mongodb, and JavaScript.

The feature's code consist mainly from four files:

1) **twitter_stream.py:** The role of this code is to get the tweets from Twitter using Twitter Streaming API. Tweepy library used to get connected with Twitter API. The tweets comes in JSON-file format. Therefore, it saved in the Mongodb using pymongo library.
2) **data.py:** The role of this code is to push the data (tweets) from mongodb to the javaScript Google Map code. The code mainly using Flask library for pushing the data. Also using Redis server to allow the special Flask port (5000) to be live in the machine.
3) **data.js:** The role of this code is reading the data coming from 'data.py' and configure the

Google Map.

4) **map.html:** This code represent the HTML web page that shows the final map using all the previous codes.

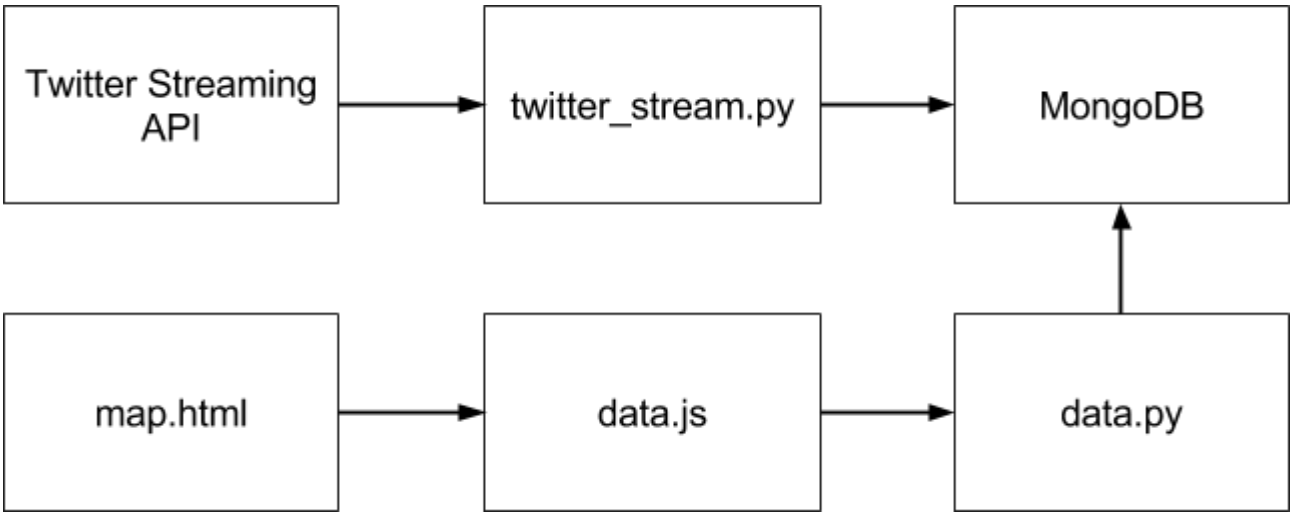

Figure 1Interactive Map Simple Arhitecture

3. PerCapita HeatMap

Although currently, at the time of the first Demo it is not yet fully integrated with the twitter data retrieval process, in the future, The technical Documentation for the Per Capita Heat Map will be identical to the Interactive Heat Map, with the addition of a weight function, based on population density, which will be applied to the data immediately before the data is displayed in the heat map.

As we are progressing through, it is becoming more apparent that we will most likely apply the "per capita weighting" concept across the many aspects of the project, instead of making this a standalone element. We may consider making a population weighted view of data as the default for our features. In this case, every feature will, at a minimum, at least have the option to convert its data into data weighted by population. In this case, each feature will have the option to "transform" the data at the moment that the "cleaned-up" data is retrieved from our database,

by the feature that is using the data.

4. CalorieMeter

The following Python packages must be downloaded using “pip”: twitter, json, counts, prettytable. CalorieMeter, uses only one Python file that includes the necessary code to authenticate with TwitterAPI, extract targeted tweets, and analyze these extracted tweets.

```
import twitter
import json
def oauth_login():
    # XXX: Go to http://twitter.com/apps/new to create an app and get values
    # for these credentials that you'll need to provide in place of these
    # empty string values that are defined as placeholders.
    # See https://dev.twitter.com/docs/auth/oauth for more information
    # on Twitter's OAuth implementation.

    CONSUMER_KEY = 'XXX'
    CONSUMER_SECRET = 'XXX'
    OAUTH_TOKEN = 'XXX'
    OAUTH_TOKEN_SECRET = 'XXX'
    auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                               CONSUMER_KEY, CONSUMER_SECRET)
    twitter_api = twitter.Twitter(auth=auth)
    return twitter_api
# Sample usage
twitter_api = oauth_login()
# Nothing to see by displaying twitter_api except that it's now a
# defined variable
print twitter_api
###############
q = 'spent #LoseIt' # CalorieMeter intresting keywords and hashtags
count = 100
# See https://dev.twitter.com/docs/api/1.1/get/search/tweets for more details
search_results = twitter_api.search.tweets(q=q, count=count)
statuses = search_results['statuses']
# Iterate through 5 more batches of results by following the cursor
for _ in range(5):
    print "Length of statuses", len(statuses)
    try:
        next_results = search_results['search_metadata']['next_results']
    except KeyError, e: # No more results when next_results doesn't exist
        break
    # Create a dictionary from next_results, which has the following form:
    # ?max_id=313519052523986943&q=NCAA&include_entities=1
```

```
    kwargs = dict([ kv.split('=') for kv in next_results[1:].split("&") ])
    search_results = twitter_api.search.tweets(**kwargs)
    statuses += search_results['statuses']
####################
status_texts = [ status['text']
                 for status in statuses ]
screen_names = [ user_mention['screen_name']
                 for status in statuses
                     for user_mention in
status['entities']['user_mentions'] ]
hashtags = [ hashtag['text']
             for status in statuses
                 for hashtag in status['entities']['hashtags'] ]
# Compute a collection of all words from all tweets
words = [ w
          for t in status_texts
              for w in t.split() ]

# Explore the first 5 items for each...
print json.dumps(status_texts[0:5], indent=1)
print json.dumps(screen_names[0:5], indent=1)
print json.dumps(hashtags[0:5], indent=1)
print json.dumps(words[0:5], indent=1)
##########################

from collections import Counter
for item in [words, screen_names, hashtags]:
    c = Counter(item)

#########################

from prettytable import PrettyTable

for label, data in (('Word', words),
                    ('Screen Name', screen_names),
                    ('Hashtag', hashtags)):
    pt = PrettyTable(field_names=[label, 'Count'])
    c = Counter(data)
    [ pt.add_row(kv) for kv in c.most_common()[:10] ]
    pt.align[label], pt.align['Count'] = 'l', 'r' # Set column alignment
    print pt
```

The results of this code should simply display the following:

`python CalorieMeter.py` (This command compiles the code)

```
<twitter.api.Twitter object at 0x7fc81d2f50d0>
Length of statuses 100
Length of statuses 200
```

```
Length of statuses 200
[
 "I spent 15 minutes boxing. 248 calories burned. #LoseIt",
 "I spent 2 hours cleaning. 331 calories burned. #LoseIt",
 "I spent 2 hours cleaning. 315 calories burned. #LoseIt",
 "I spent 20 minutes rowing a rowing machine. 155 calories burned. #LoseIt",
 "I spent 35 minutes riding a stationary bike. 263 calories burned. #LoseIt"
]
```

```
+----------+-------+
| Word     | Count |
+----------+-------+
| calories |   200 |
| burned.  |   200 |
| I        |   200 |
| spent    |   200 |
| #LoseIt  |   200 |
| minutes  |   163 |
| hour     |    57 |
| 1        |    57 |
| doing    |    50 |
| and      |    42 |
+----------+-------+
```

5. Play

The game feature of this system, as of now, consists of the Game class and the Play class. The Game Class is responsible to facilitate the pointing system of the feature. The Play class facilitates the connection between the Game Class and the MongoDB database used.

To run its unit test of the Game class include in a common folder files gameTest.py and game.py and in a terminal type:

```
python gameTest.py
```