LIVE TWEET MAP WITH SENTIMENTAL ANALYSIS

A Master’s Project
Presented to
Department of Computer and Information Sciences
SUNY Polytechnic Institute
Utica, New York

In Partial Fulfilment
Of the Requirements for the
Master of Science Degree

by

Rohila Kotrika

May 2016

© Rohila Kotrika 2016
LIVE TWEET MAP WITH SENTIMENTAL ANALYSIS

Except where reference is made to the work of others, the work described here is my own or was done in collaboration with my advisor and/or the members of the advisory committee. Further, the content of this work is truthful in regards to my own work and the portrayal of other’s work. This work, I claim, does not include proprietary or classified information.

_________________________
Rohila Kotrika
LIVE TWEET MAP WITH SENTIMENTAL ANALYSIS

Master of Science Project in Computer and Information Sciences
Department of Computer Sciences
SUNY Polytechnic Institute

Approved and recommended for acceptance as a project in partial fulfilment of the requirements for the degree of Master of Science in Computer and Information Sciences

May 10, 2016
DATE

Saumendra Sengupta, Ph.D (Advisor)

Chen-Fu Chiang, Ph.D.

Bruno Andriamanalimanana, Ph.D.
Abstract

This project basically aims to build a system for the real-time analysis of the trends and public views around the whole world by storing and analyzing the stream of tweets from the Twitter live API which produces a huge amount of data [36]. The tweets, tweet ID, time and other relevant elements are stored into a database and are represented in a map that is being updated in near real time with the help of Google map API [38]. This project also aims to achieve the sentimental analysis of the tweets by sending the tweets to the natural language processing API [39] which in turn processes the tweets using the natural language processing and gives a result if those tweets are positive, negative or neutral in nature. The map clusters tweet as to show where people are tweeting most from according to the sample tweets we get from the streaming API [35].

These clusters will be shown in different colors according to the sentimental evaluation we receive from the sentiment API by Vivek Narayanan which works by examining individual words and short sequences of words (n-grams) and comparing them with a probability model. The probability model is built on a pre labeled test set of IMDb movie reviews. It can also detect negations in phrases, i.e., the phrase “not bad” will be classified as positive despite having two individual words with a negative sentiment. The web service uses a co routine server based on event, so that the trained database can be loaded into shared memory for all requests, which makes it quite scalable and fast. The API is specified here [16], it supports batch calls so that network latency isn’t the main bottleneck. For Instance, if a tweet is negative in evaluation then it is shown in a red color marker on the map, green for positive and grey for the neutral. This analytic will also demonstrate the heat map for all the tweets that are stored in the database which gives a satisfying answer demonstrating from which part of the world are most of the tweets from.

In this project we create a dynamic web application with the target runtime environment as Apache Tomcat Server [37]. The server will also be initialized with the context listener which starts running the code to get the tweets into the database till the server is stopped. The most popular trends among worldwide and citywide would be provided in a drop down to be selected from which gives a clear perspective on how each trend behaves. It also offers the public, the media, politicians and scholars a new and timely perspective on the dynamics of the world wide trends and public opinion.
7 Analyzing Twitter Data

8 Visualizing Twitter Data

8.1 Visualizing Geospatial Information ................................................. 29
8.2 Visualizing Geo-Spatial Heat Maps .................................................. 30
8.3 Visualizing Twitter Trends ................................................................. 31

9 SQL Tables

9.1 Tweets Table .................................................................................. 34
9.2 Trends Table .................................................................................. 35

10 Conclusion and Future Achievements

Appendices

A1. Code .................................................................................................. 37

References .............................................................................................. 67
List of Figures

Figure 1: The Process of Obtaining Tweet Information from Twitter 17
Figure 2: Tweet Map Displaying Only Positive Tweets by using filter 26
Figure 3: Tweet map Displaying only negative tweets by using filter 27
Figure 4: Tweet Map displaying only neutral tweets by using filter 28
Figure 5: Tweet map with Positive, Negative and Neutral Tweets 31
Figure 6: Heat Map and Live Tweets Filtered by the Sentiment of the Tweet 32

List of Tables

Table 1: Tweets Table Containing all the information regarding a tweet 35
Table 2: Trends Table Containing the Top 10 Trends Worldwide and in NYC. 36
Chapter 1

Introduction

1.1 Background

We have decided to work with twitter because we feel it is as a better approximation of public sentiment as opposed to conventional internet articles and web blogs. The main reason is that the amount of relevant data is much larger for twitter when compared to traditional blogging sites. Furthermore, the reception on twitter is more prompt and also more general because the number of users who tweet is comparatively more than those who write web blogs on a daily basis. Sentiment analysis of public is critical in macro-scale socioeconomic phenomena like predicting the stock market rate of a particular firm. This could be done by analyzing overall public sentiment towards that firm with respect to time and using economics tools for finding the correlation between public sentiment and the firm’s stock market value. Firms can also estimate how well their product is responding in the market, which areas of the market is it having a favorable response and in which a negative response (since twitter allows us to download stream of geo-tagged tweets for particular locations). If firms can get this information they can analyze the reasons behind geographically differentiated response, and so they can market their product in a more optimized manner by looking for appropriate solutions like creating suitable market segments. Predicting the results of popular political elections and polls is also an emerging application to sentiment analysis. One such study was conducted by Tumasjan et al. in Germany for predicting the outcome of federal elections in which concluded that twitter is a good reflection of offline sentiment [1].

1.2 Motivation

This project of analyzing sentiments of tweets comes under the domain of “Pattern classification” and “Data Mining”. Both of these terms are very closely related and intertwined, and they can be formally defined as a process of discovering “useful” patterns in large set of data, either automatically (unsupervised) or semi- automatically (supervised). The project would heavily rely on techniques of “Natural Language Processing” in extracting significant patterns and features from the large data set of tweets and on “Machine Learning” techniques for accurately classifying individual unlabeled data samples (tweets) according to whichever pattern model best describes them. The features that can be used for modeling patterns and classification can be divided into two main groups: formal language based and informal blogging based. Language based features are those that deal with formal linguistics and include prior sentiment polarity of individual words.
and phrases, and parts of speech tagging of the sentence. Prior sentiment polarity means that some words and phrases have a natural innate tendency for expressing particular and specific sentiments in general. For example, the word “excellent” has a strong positive connotation while the word “evil” possesses a strong negative connotation. So whenever a word with positive connotation is used in a sentence, chances are that the entire sentence would be expressing a positive sentiment. Parts of Speech tagging, on the other hand, is a syntactical approach to the problem [2]. It means to automatically identify which part of speech each individual word of a sentence belongs to: noun, pronoun, adverb, adjective, verb, interjection, etc. Patterns can be extracted from analyzing the frequency distribution of these parts of speech (either individually or collectively with some other part of speech) in a particular class of labeled tweets [3]. Twitter based features are more informal and relate with how people express themselves on online social platforms and compress their sentiments in the limited space of 140 characters offered by twitter. They include twitter hashtags, retweets, word capitalization, word lengthening, question marks, presence of url in tweets, exclamation marks, internet emoticons and internet shorthand slangs.

1.3 What are the Issues?

Like most scientific methods sentiment analysis is not without its problems. Sentiment analysis is a very subjective method of classification and if there are more than one observer to the test, there will more than likely be differences in opinion.

This is actually problem most often encountered with sentiment analysis. Interpreting the mood of a subject may vary from one person to another; a problem made even harder by the format the subject may be analyzed in. Just how easy would it be to determine the mood of a person about a certain subject if that person only has 140 characters or less to express it in their Twitter account?

There are also technical challenges to sentiment analysis. If I were writing a response that said “An IBM iSeries System Storage DS5020 Express is the equivalent to the x450 IBM server.” Most observers without a heavy technical background or a lot of investigation will not know if that is a complimentary comparison or a negative, so that sentence will also be a problem.

Some of the main issues faced by sentimental analysis of tweets are:

**Named Entity Recognition** - What is the person actually talking about, e.g. is 300 Spartans a group of Greeks or a movie?

**Anaphora Resolution** - the problem of resolving what a pronoun, or a noun phrase refers to. "We watched the movie and went to dinner; it was awful." What does "It" refer to?

**Parsing** - What is the subject and object of the sentence, which one does the verb and/or adjective actually refer to? And abbreviations, lack of capitals, poor spelling, poor punctuation, poor grammar.
Sarcasm - If you don't know the author you have no idea whether 'bad' means bad or good.

It also raises the question if the tweets generated are by a human or bot which has been created to crowd the tweets with wrong information which may affect the public opinion. This issue can be handled by introducing an API in the code where we are storing the tweets to check the user of the tweet generated if it’s a human or bot [32]. This API takes the username of the person who tweeted and gives a response if it is a bot or a human [41].

1.4 Contribution to the knowledge

A user’s perception of a product or service is extremely valuable information to many companies and organizations. From the knowledge gained from an analysis such as this a specific company like a newspaper or a product company can identify issues with their products, spot trends before their competitors, create improved communications with their target audience, and gain valuable insight into how effective their marketing campaigns were [6]. Through this knowledge companies gain valuable feedback which allows them to further develop the next generation of their product.

In the context of the sentiment analysis being carried out for this application, the results will allow the producers of the show to gain insight into how each episode is being perceived by the viewer. This is very valuable information as viewers are uploading their expectations, opinions and views on the television program before, during and after it is aired. This really revolutionizes the feedback process, an application such as this has the potential to analyses the sentiment in real time giving the producers immediate feedback on how the program is being held in the eyes of its audience [7]. Such an application could be expanded to use clustering algorithms to give insight into particular scenes or characters. From an academic perspective it was felt that there were no new findings added to the knowledge base of natural language processing or sentiment analysis.

This visualization of the trends and sentiment of the tweets geographically gives insight for anyone or any company trying to obtain the information of the usage or the review particular to each city, state or a country.

For instance, we noticed that users generally use our website for specific types of keywords which can divided into a couple of distinct classes, namely: politics/politicians, celebrities, products/brands, sports/sportsmen, media/movies/music. So we can attempt to perform separate sentiment analysis on tweets that only belong to one of these classes (i.e. the training data would not be general but specific to one of these categories) and compare the results we get if we apply general sentiment analysis on it instead. Even the next presidential candidate can be predicated by the twitter’s sentimental analysis on each candidate which give us an idea to extend this project for future purposes.
1.5 Organization of the report

Chapter 2 will be the aims and the objectives of the research paper. The technical requirements and the general notion will be originating from the clients or user’s point of view. The normal functionalities will be made and recorded.

Chapter 3 will be dealing with representing the limitations of the prior projects which were done on this topic and also some related work to this project which is been done or being done. This section mostly focuses how we will be able to overcome the limitations of prior projects.

Chapter 4 will deal with how our proposed model will be connecting to the twitter API. Here It is explained how we are able to access twitter data with authentication, how our proposed model is able to collect the user’s information regarding a particular tweet and also the trends in given places.

Chapter 5 deals with the concept of searching and identifying the right group of tweets which are needed in this project. For example, we will be able to store the tweets which are in English because our sentimental analysis works only on English text.

Chapter 6 will be aiming to represent the process storing of the tweets in the back end database server. Here the database server we use is MySQL and in this section we will be able to learn how to install this server on our system and store the tweets and trends in structured tables.

Chapter 7 will undertake the evaluation of the sentiment of the tweet collected by the application. This section will be concentrating on how the text of the tweet is sent to the sentimental analysis API and how we are storing the sentiment of the tweet in our database.

Chapter 8 will be focusing on how to visualize the tweets that are stored in the backend MySQL server. This section deals with two main visualizations that is representing each and individual tweet according to its sentiment and also as a heat map.

Chapter 9 focuses on explaining the two tables used to store the data from twitter. The tweets table contains all the information regarding the user particular to each tweet and also the top ten trends among twitter worldwide and only in New York city.

Chapter 10 Concludes this project thesis with a small summary of all the findings and a brief discussion on how this project can be extended in the future for more purposes and in which prospective fields this project might be used.
Chapter 2

Aims and Objectives

As already mentioned in the abstract submitted for this project:

“This project basically aims to build a system for the real-time analysis of the trends and public views around the whole world by storing and analyzing the stream of tweets from the Twitter live API which produces a huge amount of data. The tweets, tweet ID, time and other relevant elements are stored into a database and are represented in a map that is being updated in near real time with the help of Google maps API.

The map clusters tweet as to show where people are tweeting most from according to the sample tweets we get from the streaming API. These clusters will be shown in different colors according to the sentimental evaluation we receive from the Alchemy API. For Instance, if a tweet is negative in evaluation then it is shown in a red color marker on the map, green for positive and grey for the neutral. This analytic will also demonstrate the heat map for all the tweets that are stored in the database which gives a satisfying answer demonstrating from which part of the world are most of the tweets from.”

2.1 Aim

The aim of this project is to create a web application that will will focus on performing a sentiment analysis on a specific sentiment or a trend or both which will be chosen at the time of visualization which depicts the public opinion which is being mined predominantly from the famous social media site twitter. This project also has a scope for analyzing the popularity of candidate or a trend in a way of letting us know if the public has a good or bad opinion. This data can also be analyzed so that the behavior of the people around the world can be known. The main goal of this sentiment analysis is to discover how people perceive the chosen candidate or a product. The opinions that are mined will be classified into three categories positive, neutral and negative. An analysis will then be performed on the classified data to see what percentage of the population sample fall into each category and it is visualized geographically.

2.2 Objectives

From the above, the primary objective is to be able to develop a dynamic web application for visualizing the tweets in near real time with the help of google maps API. The second primary objective is to be able to analyze sentiment of each tweet before storing it on the server and also show the tweets on the map depending on their polarity of the sentiment. The secondary
objectives of the project are to be able to show the heat map of all the data around the world and also to show the top ten trending topics worldwide and New York city. The secondary objectives also include displaying the user screen name, user profile image and the text of the tweet he posted in the info window of the marker.

**Importance of this area**

Sentiment analysis and Natural Language processing are very important area at the moment; there is a shortage of people skills in this area. There is a massive amount of information being uploaded to the internet daily on social media websites and blogs that computers cannot understand. Traditionally it was not possible to process such large amounts of data, but with computer performance following the projections of Moore’s law and the introduction of distributed computing (e.g. Hadoop) large data sets can now be processed with relative ease [8].

![Figure 1: Proposed Model for Our Objectives](image)
Chapter 3

Literature Review

3.1 Limitations of prior projects

Sentiment analysis in the domain of micro-blogging is a relatively new research topic so there is still a lot of room for further research in this area. Decent amount of related prior work has been done on sentiment analysis of user reviews [x], documents, web blogs/articles and general phrase level sentiment analysis. These are different from twitter because of the limit of 140 characters per tweet which forces the user to express opinion compressed in very short text [13]. The best results reached in sentiment classification use supervised learning techniques such as Naive Bayes and Support Vector Machines, but the manual labelling required for the supervised approach is very expensive. Some work has been done on unsupervised and semi-supervised approaches, and there is a lot of room for improvement. Various researchers who are testing new features and classification techniques often just compare their results to base-line performance. There is a need of proper and formal comparisons between these results arrived through different features and classification techniques in order to select the best features and most efficient classification techniques for particular applications. Most of the pervious projects don’t give an option of visualizing the distribution of the tweets geographically using a heat map and also using a normal map with markers to indicate the place of the person tweeting.

3.2 Related Work

The bag-of-words model is one of the most widely used feature model for almost all text classification tasks due to its simplicity coupled with good performance [42]. This model represents the text as a bag or collection of individual words with no link or dependence of one word with the other, i.e. it completely disregards grammar and order of words within the text.

This model is also very popular in sentiment analysis and has been used by various researchers. The simplest way to incorporate this model in our classifier is by using unigrams as features. Generally speaking, n-grams is a contiguous sequence of “n” words in our text, which is completely independent of any other words or grams in the text [42]. So unigrams are nothing but just a collection of individual words in the text to be classified, and we assume that the probability of occurrence of one word will not be affected by the presence or absence of any other word in the text.
This is a very simplifying assumption but it has been shown to provide rather good performance [14]. One simple way to use unigrams as features is to assign them with a certain prior polarity, and take the average of the overall polarity of the text, where the overall polarity of the text could simply be calculated by summing the prior polarities of individual unigrams. Prior polarity of the word would be positive if the word is generally used as an indication of positivity, for example the word “sweet”; while it would be negative if the word is generally associated with negative connotations, for example “evil”. There can also be degrees of polarity in the model, which means how much indicative is that word for that particular class. A word like “awesome” would probably have strong subjective polarity along with positivity, while the word “decent” would although have positive prior polarity but probably with weak subjectivity [33].
Chapter 4

Connecting to Twitter

Twitter’s popularity as an information source has led to the development of applications and research in various domains. Humanitarian Assistance and Disaster Relief is one domain where information from Twitter is used to provide situational awareness to a crisis situation. Researchers have used Twitter to predict the occurrence of earthquakes and identify relevant users to follow to obtain disaster related information. Studies of Twitter’s use in disasters include regions such as China and Chile. While a sampled view of Twitter is easily obtained through the APIs discussed in this book, the full view is difficult to obtain [5]. The APIs only grant us access to a 1% sample of the Twitter data, and concerns about the sampling strategy and the quality of Twitter data obtained via the API. This study indicates that care must be taken while constructing the queries used to collect data from the Streaming Users on Twitter generate over 400 million Tweets every day. Some of these Tweets are available to researchers and practitioners through public APIs at no cost. In this chapter we will learn how to extract the following types of information from Twitter:

- Information about a user,
- A user’s network consisting of his connections,
- Tweets published by a user, and
- Search results on Twitter.

APIs to access Twitter data can be classified into two types based on their design and access method:

- REST APIs are based on the REST architecture2 now popularly used for designing web APIs. These APIs use the pull strategy for data retrieval. To collect information a user must explicitly request it.

- Streaming APIs provides a continuous stream of public information from Twitter. These APIs use the push strategy for data retrieval. Once a request for information is made, the Streaming APIs provide a continuous stream of updates with no further input from the user [12]. They have different capabilities and limitations with respect to what and how much information can be retrieved. The Streaming API has three types of endpoints:

  • Public streams: These are streams containing the public tweets on Twitter.
  • User streams: These are single-user streams, with to all the Tweets of a User.

16
• Site streams: These are multi-user streams and intended for applications which access Tweets from multiple users.

4.1 Introduction to Open Authentication (OAuth)

Open Authentication (OAuth) is an open standard for authentication, adopted by Twitter to provide access to protected information. Passwords are highly vulnerable to theft and OAuth provides a safer alternative to traditional authentication approaches using a three-way handshake [9]. It also improves the confidence of the user in the application as the user’s password for his Twitter account is never shared with third-party applications. The authentication of API requests on Twitter is carried out using OAuth. Figure below summarizes the steps involved in using OAuth to access Twitter API. Twitter APIs can only be accessed by applications. Below we detail the steps for making an API call from a Twitter application using OAuth [9]:

1. Applications are also known as consumers and all applications are required to register themselves with Twitter. Through this process the application is issued a consumer key and secret which the application must use to authenticate itself to Twitter.

2. The application uses the consumer key and secret to create a unique Twitter link to which a user is redirected for authentication. The user authorizes the application by authenticating himself to Twitter. Twitter verifies the user’s identity and issues a OAuth verifier also called a PIN.

3. The user provides this PIN to the application. The application uses the PIN to request an “Access Token” and “Access Secret” unique to the user.

4. Using the “Access Token” and “Access Secret”, the application authenticates the user on Twitter and issues API calls on behalf of the user. The “Access Token” and “Access Secret” for a user do not change and can be cached by the application for future requests. Thus, this process only needs to be performed once, and it can be easily accomplished using the method GetUserAccessKeySecret.
4.2 Collecting a User’s Information

On Twitter, users create profiles to describe themselves to other users on Twitter. A user’s profile is a rich source of information about him. A user’s profile consists of all the information which is needed to profile the users depending on their tweets and the information. Using the OAuth, we are able to access the twitter’s public API which in turn gives us a stream of tweets which are only a sample of the total tweets which are being posted at that moment [10]. The important information we are storing from a user’s profile with the help of his tweets are the user’s screen name, user’s twitter Id, tweet Id, tweet, the time tweet got created, location the user tweeted the tweet from i.e. latitude and longitude. For visualizing purposes, we are also getting the profile image URL of the user. Here with the help of twitter Id we can also collect the network and followers of the user. If example Bob has 4 friends, then all the 4 friends come under the network of Bob’s social network [32].
4.3 Collecting a User’s Tweets

A Twitter user’s Tweets are also known as status messages. A Tweet can be at most 140 characters in length. Tweets can be published using a wide range of mobile and desktop clients and through the use of Twitter API. A special kind of Tweet is the retweet, which is created when one user reposts the Tweet of another [12]. A user’s Tweets can be retrieved using both the REST API and the Streaming API. We can access a user’s Tweets by using statuses/usertimeline8 from the REST APIs. Using this API, one can retrieve 3,200 of the most recent Tweets published by a user including retweets. The API returns Twitter “Tweet” objects. The process to access this API can be found in the GetStatuses method. Key Parameters: In each page, we can retrieve 200 Tweets of a user. The parameter max id is used to paginate through the Tweets of a user. To retrieve the next page, we use the ID of the oldest Tweet in the list as the value of this parameter in the subsequent request. Then, the API will retrieve only those Tweets whose IDs are below the supplied value [23].

Rate Limit: An application is allowed 300 requests within a rate limit window and up to 180 requests can be made using the credentials of a user.

STREAMING API:

Specifically, the statuses/filter API provides a constant stream of public Tweets published by a user. Using the method CreateStreamingConnection, we can create a POST request to the API and fetch the search results as a stream.

Key Parameters: The follow10 parameter can be used to specify the userids of 5,000 users as a comma.

Rate Limit: Rate limiting works differently in the Streaming API. In each connection an application is allowed to submit up to 5,000 Twitter user ids. Only public Tweets published by the user can be captured using this API.
Chapter 5
Collecting Search Results

Searching on Twitter is facilitated through the use of parameters. Acceptable parameter values for search include keywords, hashtags, phrases, geographic regions, and usernames or user ids. Twitter search is quite powerful and is accessible by both the REST and the Streaming APIs. There are certain subtle differences when using each API to retrieve search results [11].

**REST API:**

Twitter provides the search/tweets API to facilitate searching the Tweets. The search API takes words as queries and multiple queries can be combined as a comma separated list. Tweets from the previous 10 days can be searched using this API. Requests to the API can be made using the method GetSearch Results. Input to the function is a keyword or a list of keywords in the form of an OR query. The function returns an array of Tweet objects [32].

**Key Parameters:** Result type parameter can be used to select between the top ranked Tweets, the latest Tweets, or a combination of the two types of search results matching the query. The parameters max id and since id can be used to paginate through the results, as in the previous API discussions.

**Rate Limit:** An application can make a total of 450 requests and up to 180 requests from a single authenticated user within a rate limit window.

**Streaming API:**

Using the Streaming API, we can search for keywords, hashtags, user ids, and geographic bounding boxes simultaneously. The filter API facilitates this search and provides a continuous stream of Tweets matching the search criteria. POST method is preferred while creating this request because when using the GET method to retrieve the results, long URLs might be truncated. In method ProcessTwitterStream, we show how the incoming stream is processed. The input is read in the form of a continuous stream and each Tweet is written to a file periodically. This behavior can be modified as per the requirement of the application, such as storing and indexing the Tweets in a database. More discussion on the storage and indexing of Tweets will follow in [30].

**Key Parameters:** There are three key parameters:

- **follow:** a comma-separated list of user ids to follow. Twitter returns all of their public Tweets in the stream.
• **track:** a comma-separated list of keywords to track. Multiple keywords are provided as a comma separated list.

• **locations:** a comma-separated list of geographic bounding box containing the coordinates of the southwest point and the northeast point as (longitude, latitude)

• **Rate Limit:** Streaming APIs limit the number of parameters which can be supplied in one request. Up to 400 keywords, 25 geographic bounding boxes and 5,000 user ids can be provided in one request. In addition, the API returns all matching documents up to a volume equal to the streaming cap. This cap is currently set to 1% of the total current volume of Tweets published on Twitter [21].

### 5.1 Strategies to Identify the location of a Tweet

Location information on Twitter is available from two different sources:

• **Geotagging information:** Users can optionally choose to provide location information for the Tweets they publish. This information can be highly accurate if the Tweet was published using a smartphone with GPS capabilities.

• **Profile of the user:** User location can be extracted from the location field in the user’s profile. The information in the location field itself can be extracted using the APIs discussed above. Approximately 1% of all Tweets published on Twitter are geolocated. This is a very small portion of the Tweets, and it is often necessary to use the profile information to determine the Tweet’s location. This information can be used in different visualizations. The location string obtained from the user’s profile must first be translated into geographic coordinates. Typically, a gazetteer is used to perform this task [21]. A gazetteer takes a location string as input, and returns the coordinates of the location that best correspond to the string. The granularity of the location is generally coarse. For example, in the case of large regions, such as cities, this is usually the center of the city. There are several online gazetteers which provide this service, including BingTM, GoogleTM, and MapQuestTM.

In our example, we will use the Nominatimservice from MapQuest11 to demonstrate this process. A summary of the method TranslateLoc is provided, which is defined in the class LocationTranslateExample [29]. The response is provided in JSON, from which the coordinates can be easily extracted. If the service is unable to find a match, it will return (0,0) as the coordinates.
Chapter 6

Storing Twitter Data

In the previous chapter, we covered data collection methodologies. Using these methods, one can quickly amass a large volume of Tweets, Tweeters, and network information. Managing even a moderately-sized dataset is cumbersome when storing data in a text-based archive, and this solution will not give the performance needed for a real-time application. In this chapter we present some common storage methodologies for Twitter data using SQL [27]. There are several SQL implementations. In this project we choose MySQL as an example SQL implementation. We choose it for its adherence to the following principles:

Structure Oriented Storage: MySQL stores its data in rows and columns. This makes it very easy to store raw data from Twitter’s APIs.

Index Support: MySQL allows for indexes on any field, which makes it easy to create indexes optimized for your application.

Speed: Studies shows a brief comparison of query speed between the other models and MySQL which are in the favor of MySQL.

User Friendly: MySQL is very easy to install on home and personal computers which makes it accessible to everyone. The MySQL workbench acts as a GUI for the MySQL server which makes it very easy to view the data, change the constraints and alter the data.

6.1 Setting up MYSQL

6.1.1 Installing MYSQL on Windows

The simplest and recommended method is to download MySQL Installer (for Windows) and let it install and configure all of the MySQL products on your system [32]. Here is how:

• Download MySQL Installer from http://dev.mysql.com/downloads/installer/ and execute it.

Note: Unlike the standard MySQL Installer, the smaller version does not bundle any MySQL applications but it will download the MySQL products you choose to install.
• Choose the appropriate Setup Type for your system. Typically, you will choose Developer Default to install MySQL server and other MySQL tools related to MySQL development, helpful tools like MySQL Workbench. Or, choose the Custom setup type to manually select your desired MySQL products. Multiple versions of MySQL server can exist on a single system. You can choose one or multiple

• Complete the installation process by following the MySQL Installation wizard’s instructions. This will install several MySQL products and start the MySQL server.

• MySQL is now installed. You probably configured MySQL as a service that will automatically start MySQL server every time you restart your system.

Note: You probably also installed other helpful MySQL products like MySQL Workbench and MySQL Notifier on your system. Consider loading MySQL Workbench to check your new MySQL server connection, and “MySQL Notifier” to view the connections status. By default, these two programs automatically start after installing MySQL. This process also installs the MySQL Installer application on your system, and later you can use MySQL Installer to upgrade or reconfigure your MySQL.

The java code in the below page shows the DbConnector class which is put along with the other classes in the package. For this code to execute properly the JDBC driver for MySQL must be installed with the help of MySQL installer. This class imports all the packages required for it to create a connection and then the url of the MySQL database, username and password are required to connect to the MySQL server. These fields are used to create a connection and can be opened and closed whenever with the help of conn.open() and conn.close() statements [23].
6.2 Java Code for Connecting to MYSQL

```java
package src.db;
import java.sql.Connection;
import java.sql.DriverManager;

public class DbConnector {
    public static Connection getConnection() {
        Connection conn = null;
        try {
            String db_url = "jdbc:mysql://localhost:3306/test";
            String db_user = "root";
            String db_pass = "8977";
            Class.forName("com.mysql.jdbc.Driver");
            conn = (Connection) DriverManager.getConnection(db_url, db_user, db_pass);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            return conn;
        }
        return conn;
    }
}
```
Chapter 7

Analyzing Twitter Data

So far we have discussed the collection and management of a large set of Tweets. It is time to put these Tweets to work to gain information about the data we have collected. This chapter focuses on two key aspects of Twitter data for data analysis: This is done by using the sentiment tool created by Author Vivek Narayan which works by examining individual words and short sequences of words (n-grams) and comparing them with a probability model. The probability model is built on a pre-labeled test set of IMDb movie reviews. It can also detect negations in phrases, i.e., the phrase “not bad” will be classified as positive despite having two individual words with a negative sentiment [15]. The web service uses a co-routine server based on event, so that the trained database can be loaded into shared memory for all requests, which makes it quite scalable and fast. The API is specified here; it supports batch calls so that network latency isn’t the main bottleneck. so that we can take average opinion of people on the sentiment of the tweet and in this way the noise and inaccuracies in labelling can be minimized [17]. Generally speaking, the more copies of labels we can get the better it is, but we have to keep the cost of labelling in our mind, hence we reached at the reasonable figure of three.

The API labels the tweets in three classes according to sentiments expressed/observed in the tweets [18]:

1) Positive
2) Negative
3) Neutral

These each positive, negative and neutral sentiments are explained with an examples in the below. These examples give an idea how the tweet’s text is categorized into these categories.

Positive:
If the entire tweet has a positive/happy(excited/joyful attitude or if something is mentioned with positive connotations. Also if more than one sentiment is expressed in the tweet but the positive sentiment is more dominant.

Example: “4 more years of being in shithole Australia then I move to the USA! :D”.
Negative:

If the entire tweet has a negative/sad/displeased attitude or if something is mentioned with negative connotations. Also if more than one sentiment is expressed in the tweet but the negative sentiment is more dominant.

Example: “I want an android now this iPhone is boring :S”.
Neutral:

If the creator of tweet expresses no personal sentiment/opinion in the tweet and merely transmits information. Advertisements of different products would be labelled under this category.

Example: “US House Speaker vows to stop Obama contraceptive rule...  http://t.co/cyEWqK1E”.
FIGURE 5: TWEET MAP DISPLAYING ONLY NEUTRAL TWEETS BY USING FILTER

<Blank>:

Leave the tweet unlabeled if it belongs to some language other than English so that it is ignored in the training data.

Sentiment API:

To use the API, send a HTTP POST request to http://sentiment.vivekn.com/api/text/ with the key containing the text you want to classify [16]

You will receive a JSON response of the form:

```json
{ "result": { "sentiment": "Positive", "confidence": 3.422451 } }
```

A batch API has been added as the bottleneck for most of you will be in the network latency. To use it send a POST request with a JSON array of the texts you want to classify to http://sentiment.vivekn.com/api/batch/. You will receive another JSON array of responses (similar to the one above) corresponding to the same order as in the request. The maximum content size can be 1 MB per request.
Chapter 8

Visualizing Twitter Data

8.1 Visualizing Geospatial Information

Geo-spatial visualization can help us answer the following two questions:

• Where are events occurring?
• Where are new events likely to occur? Location information of a Tweet can be identified using two techniques as explained in Chapter 2:
  • Accurately through the geotagging feature available on Twitter.
  • Approximately using the location in the user’s profile.

The location information is typically used to gain insight into the prominent locations discussing an event. Maps are an obvious choice to visualize location information. In this section, we will discuss how maps can be used to effectively summarize location information and aid in the analysis of Tweets.

A first attempt at creating a map identifying Tweet locations would be to simply highlight the individual Tweet locations. Each Tweet is identified by a dot on the map, and such dots are referred to as markers [17]. Typically, the shape, color, and style of a marker can be customized to match the application requirements. Maps are rendered as a collection of images, called tiles. An example of the “dots on map” approach is presented in Figure below which shows the locations on Google Maps. As we are using google maps API to connect the tweets data we have to update the Map in the application. The sentiment of the tweets also is used in visualizing the data, If the tweet is positive then it is visualized with a green dot, if it’s a negative tweet then it is visualized using a red dot and grey dot for a tweet which is sentimentally neutral [20]. If you hover over the markers(dots) then you can see the profile image of the user, tweet and the sentiment.
The “dots on map” approach is not scalable and can be unreadable when there are too many markers. Additionally, when multiple Tweets originate from a very small region, the map in the above Figure can mislead readers into thinking that there are fewer than actual markers due to marker overlap [28]. One approach to overcome this problem is to use heat maps. In a geo-spatial visualization, we want to quickly identify regions of interest or regions of high density of Twitter users. This information for example could be used for targeted advertising as well as customer base estimation. Kernel Density Estimation is one approach to estimating the density of Tweets and creating such heat maps, which highlight regions of high density [19].

Kernel Density Estimation(KDE):

Kernel Density Estimation is a non-parametric approach to estimating the probability density function of the distribution from the observations, which in this case are Tweet locations. KDE attempts to place a kernel on each point and then sums them up to discover the overall distribution. Appropriate kernel functions can be chosen based on the task and the expected distribution of the points [19]. A smoothing parameter called bandwidth is used to decide if the learned kernel will be smooth or bumpy. Using KDE, we can generate a heat map from the Tweet locations, which is presented in Figure below. This figure clearly highlights the regions of high density and effectively
summarizes the important regions in our dataset when compared to the previous Figure 7.1. This estimate of KDE and visualizing is all done by the Google Maps where we just have to send the locations of each tweet. We are limiting the number of tweets it loads on the map to 10,000 because of the time the map takes to display them.

![FIGURE 7: HEAT MAP AND LIVE TWEETS FILTERED BY THE SENTIMENT OF THE TWEET](image)

### 8.3 Visualizing Twitter Trends

Returns the top 50 trending topics for a specific WOEID, if trending information is available for it. The response is an array of “trend” objects that encode the name of the trending topic, the query parameter that can be used to search for the topic on Twitter Search, and the Twitter Search URL. This information is cached for 5 minutes. Requesting more frequently than that will not return any more data, and will count against your rate limit usage [33]. The tweet volume (tweet_volume) for the last 24 hours is also returned for every trend. Please note that Ads API developers have an increased rate limit on this endpoint at 50 queries / 15 minutes / token.

**Resource URL**

https://api.twitter.com/1.1/trends/place.json
**Example Request**

https://api.twitter.com/1.1/trends/place.json?id=1

**Example Result**

```
[

{
  "as_of": "2012-08-24T23:25:43Z",
  "created_at": "2012-08-24T23:24:14Z",
  "locations": [
    {
      "name": "Worldwide",
      "woeid": 1
    }
  ],
  "trends": [
    {
      "tweet_volume": 3200,
      "events": null,
      "name": "#GanaPuntosSi",
      "promoted_content": null,
      "query": "%23GanaPuntosSi",
      "url": "http://twitter.com/search/?q=%23GanaPuntosSi"
    },
    {
      "tweet_volume": 4200,
      "events": null,
```

```
"name": "#WordsThatDescribeMe",
"promoted_content": null,
"query": "%23WordsThatDescribeMe",
"url": "http://twitter.com/search/?q=%23WordsThatDescribeMe"
},

{
"tweet_volume": 1200,
"events": null,
"name": "#10PersonasQueExtrañoMucho",
"promoted_content": null,
"query": "%2310PersonasQueExtra%C3%B1oMucho",
"url": "http://twitter.com/search/?q=%2310PersonasQueExtra%C3%B1oMucho"
}
Chapter 9

SQL Tables

9.1 Tweets table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>screen_name</td>
<td>char(20)</td>
</tr>
<tr>
<td>tweet_id</td>
<td>bigint(20)</td>
</tr>
<tr>
<td>tweet_text</td>
<td>varchar(200)</td>
</tr>
<tr>
<td>geo_lat</td>
<td>decimal(10,5)</td>
</tr>
<tr>
<td>geo_long</td>
<td>decimal(10,5)</td>
</tr>
<tr>
<td>profile_image_url</td>
<td>varchar(200)</td>
</tr>
<tr>
<td>created_at</td>
<td>timestamp(6)</td>
</tr>
<tr>
<td>sentiment</td>
<td>varchar(45)</td>
</tr>
</tbody>
</table>

Table 1: Tweets table containing all the information regarding a tweet
The above table gives information about the user details.

**User_id:** it gives info about the id of the user. Each and every user will have unique id.

**Screen_name:** It gives the name of the user indicated in his twitter account.

**Tweet_id:** It gives the id of the tweet the user has tweeted. Tweet_id is also unique for every single user.

**Tweet_text:** It shows the text which the user has tweeted

**Geo_lat and geo_long:** it shows the latitude and longitude of the place where the tweet came from.

**Profile_image_url:** This field contains the image URL of the user which can be shown during visualization.

**Created_at:** It gives info about when the tweet is created.

**Sentiment:** Sentiment gives details about the tweet weather it is positive, negative or neutral.

### 9.2 Trends Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>place</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>trend</td>
<td>varchar(45)</td>
</tr>
<tr>
<td>date</td>
<td>timestamp(6)</td>
</tr>
</tbody>
</table>

**TABLE 2:** Trends table containing the top 10 trends worldwide and in NYC.

In the above table “Trends” we include three columns like “place” which indicate the location of the tweet that is trending. “Trend” give information about the latest trending topic. “Date” gives info about when that trend started.
Chapter 10

Conclusion and Future Achievements

The task of sentimental analysis, especially in the domain of micro-blogging is in a state of developing and it’s incomplete which motivates us to propose some ideas which are worth exploring in the future and may reveal an increase in the performance [24]. The present model we have worked on is with the simplest ones, this can be improved in the future by adding the natural language processing in the project rather than importing the sentiment from a third party API. Even though the API we are getting the sentiment from is providing us with a weight of the sentiment along with the type of the sentiment but we are just discarding the weight because we are not visualizing any analytic based on the correctness of the sentiment. This can be improved in the future by analyzing the visualization of the tweets according to its weight which shows the correctness of the sentiment.

One potential problem with our project is that we are providing the filter for the trends by giving them manually or only the top ten trends according to that place. This section in the future can be improved by showing the trend filter in different categories and different places [26]. The trends can be first filtered by the category like music, travel or movies etc. which makes the trends even more accurate for those type of tweets.

Another potential problem with our project is that sizes of the three classes are not equal. The positive and negative tweets are around 25% each and the neutral tweets are around 50% of the total tweets [33]. This gives an idea why most of the tweets that are recorded are neutral, the sentiment tool we are using in our project is a reputed one but still there were some tweets recorded in the database which were having some special characters and some doodles which usually don’t make sense for a sentiment tool, so at these times mostly the API returns the tweet as neutral because of no negative or positive words in it. In future we should be able to discard the tweets which contains only special characters or at least make the tool in such a way that it recognizes the tweet with those type of Last but not the least, in this project we are solely focusing on general sentiment analysis which is just beginning of the work in this field. There is a lot of potential work in the field of sentimental analysis with partially known context. For example, we noticed that users generally use our website for specific types of keywords which can divided into a couple of distinct classes [25]. So we can attempt to perform separate sentiment analysis on tweets that only belong to one of these classes (i.e. the training data would not be general but specific to one of these categories) and compare the results we get if we apply general sentiment analysis on it instead.
Index.jsp: This is the code for our main page. It shows tweet map along with the trends

```java
<%@ page language="java" import="java.util.*" pageEncoding="ISO-8859-1"%>

String path = request.getContextPath();
String basePath =
request.getScheme() + "/" + request.getServerName() + ":" + request.getServerPort() + path + "/";

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no">
<meta charset="utf-8">
<title>Tweet-Map</title>
<link rel="stylesheet" type="text/css" href="css/basic.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script type="text/javascript">
var markers = [];
var map;
var filter_args = "";
function clickfunc(obj) {
var t = $(obj).text();
var filter_ag = t ;
$.ajax({
    type: "POST",
    url: "GetMarkers",
    data: {"filter": filter_ag, "sent": filter_args},
    dataType: "json",
    success: function(responseText){
        var keys=[];
        for (var i in responseText ) {
            keys.push(i);
        }

        var t = responseText[keys[0]];

        // for initializing map and for first marker
        var myLatlng = new google.maps.LatLng(51.508742,-0.120850);
        var mapOptions = {
            zoom: 4,
            center: myLatlng,
            mapTypeId:google.maps.MapTypeId.ROAD
        };

        map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
    }
});
```
addMarker(myLatLng);
console.log(keys.length);

for ( var i=0 ; i<keys.length ; i++ ) {
    var t = responseText[keys[i]];
    var innerLatLng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]) ;
    var title = " TweetID: " + t["tweet_id"] + " | Latitude: " + t["geo_lat"] + " | Longitude: " + t["geo_long"] + " | Tweet: " + t["tweet_text"] ;

    var contentString = "<img width='50' src=" + t["profile_image_url"] + ">
    + '<br>' + '<b>User Name:</b> ' + t["screen_name"] + '<br>' + '<b>Tweet:</b> ' + t["tweet_text"] + '<br>' + '<b>Sentiment:</b> ' + t["sentiment"] ;

    var senti = t["sentiment"] ;
    addMarker(innerLatLng, title, contentString, senti) ;
}

error: function(request, error, data) {
    alert("ajax error");
}

var image0 = {
    url: 'images/red.png'
}

var image1 = {
    url: 'images/green-dot.png'
}

var image2 = {
    url: 'images/grey.png'
}

function addMarker(location, ttitle, contentString, senti) {
    if (senti == "Positive") {
        image = image1 ;
    } else if (senti == "Negative") {
        image = image0 ;
    } else {
        image = image2 ;
    }

    var marker = new google.maps.Marker({
        position: location,
        map: map,
        title: ttitle,
        icon : image
    });
}
markers.push(marker);

`var infowindow = new google.maps.InfoWindow({
    content: contentString,
    disableAutoPan: true
});`

google.maps.event.addListener(marker, 'mouseover',

`function() {
    infowindow.open(map, this);
});`

google.maps.event.addListener(marker, 'mouseout',

`function() {
    infowindow.close();
});`

`var asyncRequest;`
`function start() {`
`    try {
        asyncRequest = new XMLHttpRequest();
        asyncRequest.addEventListener("readystatechange", stateChange, false);
        asyncRequest.open('GET', 'GetMarkers', true); // /Test is url to Servlet!
        asyncRequest.send(null);
    }`
`    catch(exception) {
        alert("Request failed");
    }
    }`

`function stateChange() {
    if (asyncRequest.readyState == 4 && asyncRequest.status == 200) {
        `var text = document.getElementById("panel");
        text.innerHTML = "<b><h2>Trending</h2></b> <a href="http://localhost:8080/tweetfinal/index.jsp">ALL</a><br />" + asyncRequest.responseText; // div in HTML document
    }
}

window.addEventListener("load", start(), false);

`function toggleHeatmap() {
  window.location.href = "http://localhost:8080/tweetfinal/HeatMap.jsp";
}
`
Filter the tweets with help of top 5 most used keywords.

Filter:

Category:
Heatmap.jsp: This is the code for the heat map page.

```html
<!-- Heatmap.jsp -->

<%@ page import="java.io.PrintWriter" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Heat Map</title>

<style type="text/css">
    html, body {
        height: 100%;
        margin: 0;
        padding: 0;
    }
    #map {
        height: 100%;
    }

    #floating-panel {
        position: absolute;
        top: 10px;
        left: 25%;
        z-index: 5;
        background-color: #fff;
        padding: 5px;
        border: 1px solid #999;
        text-align: center;
        font-family: 'Roboto','sans-serif';
        line-height: 30px;
        padding-left: 10px;
    }

    #floating-panel {
        background-color: #fff;
        border: 1px solid #999;
        left: 25%;
        padding: 5px;
        position: absolute;
        top: 10px;
        z-index: 5;
    }
</style>

</head>
<body>
</body>
</html>
```
<div id="floating-panel">
    <button onClick="toggleHeatmap()">Toggle Heatmap</button>
    <button onClick="changeGradient()">Change gradient</button>
    <button onClick="changeRadius()">Change radius</button>
    <button onClick="changeOpacity()">Change opacity</button>
    <button onClick="tweetMap()">Tweet Map</button>
</div>

<div id="dropDown1">
    <b>Polarity:</b> </div>

<select name="selectTrends1" id="trends1">
    <option value="all">All</option>
    <option value="Positive">Positive</option>
    <option value="Negative">Negative</option>
    <option value="Neutral">Neutral</option>
</select>

</div>

<div id="map"></div>

<script>
    var interval = 0;

    function start(){
        setTimeout(function(){
            ajax_function();
            interval = 60;
            setInterval(function(){
                ajax_function();
                }, interval * 1000);
            }, interval * 1000);
    }

    var map, heatmap;
    var markers = [];
    var filter_args = ""
    pointArray = new google.maps.MVCArray([]);
    var liveTweets = new google.maps.MVCArray();
</script>
$(document).ready(function(){
  $.ajax({
    type: "POST",
    url: "GetMarkers",
    data: {"filter": filter_args},
    dataType: "json",
    success: function(responseText){
      var keys=[];
      for (var i in responseText ) {
        keys.push(i);
      }
      var t = responseText[keys[0]];
      for ( var i=1 ; i<keys.length ; i++ ) {
        var t = responseText[keys[i]];
        pointArray.push(new google.maps.LatLng(t["geo_lat"], t["geo_long"]));
      }
    },
    error: function(request, error, data){
      alert("ajax error");
    }
  });

  var time = new Date();
  interval = 60 - time.getSeconds();
  if(interval==60)
    interval = 0;
  start();

});

$('#trends1').change(function(){
  var filter_ag = ";
  if($(this).val()!= "all"){
    var msg = $(this).val();
  }
  else{
    var msg = ";
  }
  $.ajax({
    type: "POST",
    url: "GetMarkers",
    data: {"filter": filter_args},
    dataType: "json",
    success: function(responseText){
      var keys=[];
      for (var i in responseText ) {
        keys.push(i);
      }
      var t = responseText[keys[0]];
      for ( var i=1 ; i<keys.length ; i++ ) {
        var t = responseText[keys[i]];
        pointArray.push(new google.maps.LatLng(t["geo_lat"], t["geo_long"]));
      }
    },
    error: function(request, error, data){
      alert("ajax error");
    }
  });

});
type: "POST",
url: "GetMarkers",
data: {"filter": filter_ag,"sent": msg},
dataType: "json",
success: function(responseText){
  var keys=[];
  for (var i in responseText )
  {
    keys.push(i);
  }
  var t = responseText[keys[0]];
  for ( var i=1 ; i<keys.length ; i++ )
  {
    var t = responseText[keys[i]];
    pointArray.push(new google.maps.LatLng(t["geo_lat"], t["geo_long"]));
  }
},
error: function(request, error, data)
{
  alert("ajax error");
}
});

function ajax_function()
{
  $.ajax({
    type: "GET",
    url: "GetLive",
    data: {"filter": filter_args},
dataType: "json",
success: function(responseText){
  var keys1 =[];
  for (var i in responseText )
  {
    keys1.push(i);
  }
  var t = responseText[keys1[0]];
  clearMarkers();

  // Flash a dot onto the map quickly
  var image = "images/small-dot-icon.png";

  for ( var i=1 ; i<keys1.length ; i++ )
  {
    var t = responseText[keys1[i]];
  }
  });
  });
});
var liveLocation = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);

var live = liveLocation;

(function(liveLocation){
    setTimeout(function() {
        addMarker(liveLocation);
    }, i * 500);
}(live));

error: function(request, error, data){
    alert("ajax error");
}

}

function initialize() {
    var mapOptions = {
        zoom: 2,
        center: new google.maps.LatLng(37.774546, -122.433523),
        mapTypeId: google.maps.MapTypeId.SATELLITE
    };

    map = new google.maps.Map(document.getElementById('map'),
        mapOptions);

    heatmap = new google.maps.visualization.HeatmapLayer({
        data: pointArray
    });

    heatmap.setMap(map);
    heatmap.set('radius',20);
}

function toggleHeatmap() {
    heatmap.setMap(heatmap.getMap() ? null : map);
}

function changeGradient() {
    var gradient = [
        'rgba(0, 255, 255, 0)',
        'rgba(0, 255, 255, 1)',
        'rgba(0, 191, 255, 1)',
        'rgba(0, 127, 255, 1)',
        'rgba(0, 63, 255, 1)',
        'rgba(0, 0, 255, 1)',
        'rgba(0, 0, 223, 1)',
        'rgba(0, 0, 191, 1)',
        'rgba(0, 0, 159, 1)',
        'rgba(0, 0, 127, 1)',
        'rgba(0, 0, 63, 1)',
        'rgba(0, 0, 0, 1)',
        'rgba(0, 0, 0, 0)
    ];

    heatmap.set('gradient', gradient);
    heatmap.set('radius', 20);
    heatmap.setMap(map);
}
function changeRadius() {
    heatmap.set('radius', heatmap.get('radius') ? null : 20);
}

function changeOpacity() {
    heatmap.set('opacity', heatmap.get('opacity') ? null : 0.2);
}

function tweetMap() {
    window.location.href = "http://localhost:8080/tweetfinal";
}

var imagex = "images/small-dot-icon.png";

function addMarker(m) {
    var marker = new google.maps.Marker({
        position: m,
        map: map,
        draggable: false,
        icon: imagex,
        animation: google.maps.Animation.DROP
    });
    markers.push(marker);
}

function clearMarkers() {
    for (var i = 0; i < markers.length; i++) {
        markers[i].setMap(null);
    }
    markers = [];
}

</script>
<script async defer src="https://maps.googleapis.com/maps/api/js?v=3.exp&key=AIzaSyA-NwaVaNEMv2E4NDTL6KbjzBmRhD6xNc&signed_in=true&libraries=visualization&callback=initialize"></script>
heatmap.setData(pointArray);
</body>
</html>
Basic.css

h1{
  margin: -10px -10px -10px -10px;
  font-weight: normal;
  position: center;
  font-size: 32px;
  line-height: 50px;
  background: DarkBlue;
  padding: 5px 15px;
  color: white;
  font-family: 'Muli', sans-serif;
}

h2{
  font-weight: bold;
  position: center;
  font-size: 30px;
  font-family: 'Muli', sans-serif;
  text-decoration: underline;
}

h3{
  font-weight: bold;
  position: relative;
  font-size: 20px;
  font-family: 'Muli', sans-serif;
  text-decoration: underline;
}

#map-canvas{
  width: 900px;
  height: 600px;
  margin-top: 5px;
  border-right: solid #121111;
}

#panel {
  margin-top: 0px;
  margin-bottom: 0px;
}

.canvas{
  float: left;
  margin-right: 20px;
}

.pan{
  margin-left: 30px;
}

#floating-panel {
  position: absolute;
  top: 10px;
  left: 25%;
  z-index: 5;
Basic.js:

$(document).ready(function()
{
    var markers = [];
    var map;
    var msg = "";
    var filter_args = "";
    var getdata1;
    var getdata2;

    $.ajax{
        type: "POST",
        url: "GetMarkers",
        data: {"filter": filter_args, "sent": msg},
        dataType: "json",
        success: function(responseText){
            var keys=[];
            for (var i in responseText ) {
                keys.push(i);
            }
            var t = responseText[keys[0]];
            var myLatlng = new google.maps.LatLng(51.508742,-0.120850);
            var mapOptions = {
                zoom: 4,
                center: myLatlng,
                mapTypeId:google.maps.MapTypeId.ROAD
            };
            map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
            addMarker(myLatlng);
            for ( var i=1 ; i<keys.length ; i++ )
            { 
                var t = responseText[keys[i]];
                var title = "TweetID: " + t["tweet_id"]+" | Latitude: "+t["geo_lat"]+" | Longitude: "+t["geo_long"]+" | Tweet: " +t["tweet_text"]+"<br>" +'<b>User Name:</b>'+t["screen_name"]+"<br>"+"<b>Tweet:</b>" +t["tweet_text"]+"<br>"+"<b>Sentiment:</b>" +t["sentiment"];
                var senti = t["sentiment"];
                addMarker(innerLatLng, title, contentString, senti);
            }
        },
        error: function(request, error, data){
            alert("ajax error");
        }
    });
});
$("#filter").change(function(){
    if(document.getElementById("trends1").value != "all")
    {
        var msg = document.getElementById("trends1").value ;
    }
    else{
        var msg = ";
    }
    if($(this).val()!='all')
    {
        var filter_ag = $(this).val();
        $.ajax({
            type: "POST",
            url: "GetMarkers",
            data: {"filter": filter_ag, "sent": msg},
            dataType: "json",
            success: function(responseText){
                var keys=[];
                for (var i in responseText )
                {
                    keys.push(i);
                }
                    var t = responseText[keys[0]];
                    var myLatlng = new
google.maps.LatLng(51.508742,-0.120850);
                    var mapOptions = {
                        zoom: 4,
                        center:
                        myLatlng,
                        mapTypeId:google.maps.MapTypeId.ROAD
                }; map =
            google.maps.Map(document.getElementById('map-canvas'), mapOptions);

            addMarker(myLatlng);
            for ( var i=1 ; i<keys.length ; i++ )
            { console.log("inside");
                var t = responseText[keys[i]];
                var innerLatlng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);
                var title = " TweetID: " + t["tweet_id"]+" | Latitude: "+t["geo_lat"]+" |
                Longitude: "+t["geo_long"]+" | Tweet: "+t["tweet_text"];
                var contentString = "<img width='50' src=" + t["profile_image_url"] + ">
                + ' <br>' + "User Name:" +t["screen_name"]+"<br>" + "Tweet:"+ "<b>Tweet:<b>
                " +t["tweet_text"]+"<br>"+"<b>Sentiment:<b> "+t["sentiment"];
                var senti = t["sentiment"];
    }}
    else{
        var msg = "";
    }
});
addMarker(innerLatLng, title, contentString, senti);
}

error: function(request, error, data){
    alert("ajax error");
}
}

else {

    var filter_args = "";
    $.ajax({
        type: "POST",
        url: "GetMarkers",
        data: {"filter": filter_args, "sent": msg},
        dataType: "json",
        success: function(responseText){
            var keys=[];
            for (var i in responseText ) {
                keys.push(i);
            }
            var t = responseText[keys[0]];
            var myLatlng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);
            var mapOptions = {
                zoom: 4,
                center: myLatlng,
                mapTypeId:google.maps.MapTypeId.ROAD
            };
            map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);
            addMarker(myLatlng);
            for ( var i=1 ; i<keys.length ; i++ )
            {
                console.log("inside");
                var t = responseText[keys[i]];
                var innerLatLng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);
                var title = " TweetID: " + t["tweet_id"] + " | Latitude: " + t["geo_lat"] + " | Longitude: " + t["geo_long"] + " | Tweet: " + t["tweet_text"];
                var contentString = "<img width='50' src= " + t["profile_image_url"] + ">
                + "<b>User Name:</b> " + t["screen_name"] + '<br>' + "<b>Tweet:</b> " + t["tweet_text"] + '<br>' + "<b>Sentiment:</b> " + t["sentiment"];
                var senti = t["sentiment"];
                addMarker(innerLatLng, title, contentString,senti);
            }
        },
        error: function(request, error, data){
            alert("ajax error");
        }
    });
}
var image0 = {
    url: 'images/red.png'
}

var image1 = {
    url: 'images/green-dot.png'
}

var image2 = {
    url: 'images/grey.png'
}

function addMarker(location, ttitle, contentString, senti) {
    if(senti == "Positive") {
        image = image1;
    } else if(senti == "Negative") {
        image = image0;
    } else {
        image = image2;
    }
    var marker = new google.maps.Marker({
        position: location,
        map: map,
        title: ttitle,
        icon: image
    });
    markers.push(marker);
    var infowindow = new google.maps.InfoWindow({
        content: contentString,
        disableAutoPan: true
    });
google.maps.event.addListener(marker, 'mouseover',
function() {
    infowindow.open(map, this);
});
google.maps.event.addListener(marker, 'mouseout',
function() {
    infowindow.close();
});
}

function setAllMap(map) {
    for (var i = 0; i < markers.length; i++) {
        markers[i].setMap(map);
    }
}
function showMarkers()
{
    setAllMap(map);
}

function showMap()
{
google.maps.event.addDomListener(window, 'load', getMap);
var t = "ladf";
}

$('#trends1').change(function()
{
    if($(this).val() != "all")
    {
        var msg = $(this).val();
    }

    if(document.getElementById("filter").value != "all")
    {
        var filter_ag = document.getElementById("filter").value ;
        $.ajax({
            type: "POST",
            url: "GetMarkers",
            data: {"filter": filter_ag, "sent": msg},
            dataType: "json",
            success: function(responseText){
                var keys=[];
                for (var i in responseText )
                {
                    keys.push(i);
                }

                var t = responseText[keys[0]];
                var myLatlng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);
                var mapOptions = {
                    zoom: 4,
                    center: myLatlng,
                    mapTypeId:google.maps.MapTypeId.ROAD
                };
            map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);

                addMarker(myLatlng);
                for ( var i=1 ; i<keys.length ; i++ )
                {
                    console.log("inside");
                    var t = responseText[keys[i]];
                    var innerLatLng = new google.maps.LatLng(t["geo_lat"], t["geo_long"]);
                }
            }
        });
    }
}
var title = "TweetID: " + t['tweet_id'] + " | Latitude: " + t['geo_lat'] + " | Longitude: " + t['geo_long'] + " | Tweet: " + t['tweet_text'];

var contentString = "<img width='50' src=" + t['profile_image_url'] + ">
" + '<br>' + '<b>User Name:</b> ' + t['screen_name'] + '</b><br>' + '<b>Tweet:</b> ' + t['tweet_text'] + '<br>' + '<b>Sentiment:</b> ' + t['sentiment'];

var senti = t['sentiment'];
addMarker(innerLatLng, title, contentString, senti);
}
,

error: function(request, error, data){
    alert("ajax error");
}
});

else {

var filter_args = "";
$.ajax({
    type: "POST",
    url: "GetMarkers",
    data: {"filter": filter_args, "sent": msg},
    dataType: "json",
    success: function(responseText){
        var keys=[];
        for (var i in responseText ) {
            keys.push(i);
        }
        var t = responseText[keys[0]];

        var myLatlng = new google.maps.LatLng(51.508742,-0.120850);
        var mapOptions = {
            zoom: 4,
            center: myLatlng,
            mapTypeId:google.maps.MapTypeId.ROAD
        };

        map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);

        //google.maps.event.addListener(window,'load', map);

        addMarker(myLatlng);
        for ( var i=1 ; i<keys.length ; i++ ) {
            console.log("inside");
        };
    },
    error: function(request, error, data){
        alert("ajax error");
    }
});
}
var t = responseText[keys[i]];
var innerLatLng = new google.maps.LatLng(t['geo_lat'], t['geo_long']);
var title = "TweetID: " + t['tweet_id'] + '| Latitude: " + t['geo_lat'] + " | Longitude: " + t['geo_long'] + " | Tweet: " + t['tweet_text'];
var contentString = "<img width='50' src=" + t['profile_image_url'] + ">
' + '<br>' + '<b>User Name:</b> ' + t['screen_name'] + '<br>' + '<b>Tweet:</b> ' + t['tweet_text'] + '<br>' + '<b>Sentiment:</b> ' + t['sentiment'];
var senti = t['sentiment'];
addMarker(innerLatLng, title, contentString, senti);

error: function(request, error, data){
   alert("ajax error");
};

});
});
TweetGet.jsp : This code gives information about how to store tweets in the database.

```java
package src.googlemap;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import javax.servlet.*;
import src.db.DbConnector;
import twitter4j.StallWarning;
import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
import twitter4j.StatusListener;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.conf.ConfigurationBuilder;

/**
 * This is a code example of Twitter4J Streaming API - sample method support.<p>
 * Usage: java twitter4j.examples.PrintSampleStream<br/>
 * </p>
 * @author Yusuke Yamamoto - yusuke at mac.com
 */

public final class TweetGet implements ServletContextListener {
    /**
     * Main entry of this application.
     * @param args
     */
    public static Connection con = null;
    public static Statement st = null;
    public void init() throws ServletException {
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true)
            .setOAuthConsumerKey("QMa6eYzVHeJVTqOnsg8vX86cD")
            .setOAuthConsumerSecret("54McLmSu2p0ocrWa4BTOelXtQGi9ySfL2swTEFVjB379roolOA")
            .setOAuthAccessToken("127786579-9nHeQ21j5Ye7tMjiohCee37q9Fxq2snp7c6j8")
            .setOAuthAccessTokenSecret("BkV0vJvQ9wyik9oxbRPr5IZ64xCu0vADawm1U5MK45");
        // just fill this
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
        }
    }
}
```
TwitterStream twitterStream = new TwitterStreamFactory(cb.build()).getInstance();

StatusListener listener = new StatusListener() {

    @Override
    public void onStatus(Status status) {

        if(status.getLang().equals("en")) {

            try {

                con = DbConnector.getConnection(); // to connect to the database
                st = con.createStatement();

                String qStr = status.getId()+"$"+status.getUser().getScreenName()+"$"+status.getGeoLocation().getLatitude()+"$"+status.getGeoLocation().getLongitude()+"$"+status.getUser().getId()+"$"+status.getUser().getProfileImageURL()+"$"+new Timestamp(date.getTime());

                String[] fields = qStr.split("\$");

                System.out.println("adding to database");


            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        } else {

        }

    }

    } finally{
        try{
            if(con!=null)
                con.close();
        } finally{
            try{
                if(con!=null)
                    con.close();
            } finally{
        }

    }
catch (SQLException se) {
    // se.printStackTrace();
}

else {
}

@Override
public void onDeletionNotice(StatusDeletionNotice statusDeletionNotice) {
    System.out.println("Got a status deletion notice id:" +
    statusDeletionNotice.getStatusId());
}

@Override
public void onTrackLimitationNotice(int numberOfLimitedStatuses) {
    System.out.println("Got track limitation notice:" +
    numberOfLimitedStatuses);
}

@Override
public void onScrubGeo(long userId, long upToStatusId) {
    System.out.println("Got scrub_geo event userId:" + userId + 
    upToStatusId:" + upToStatusId);
}

@Override
public void onStallWarning(StallWarning warning) {
    System.out.println("Got stall warning:" + warning);
}

@Override
public void onException(Exception ex) {
    ex.printStackTrace();
}

};
twitterStream.addListener(listener);
twitterStream.sample();

public static void StopTweets() {
    try{
        if (con != null)
            con.close();
    } catch (SQLException se) {
        // se.printStackTrace();
    }
}

@Override
public void contextDestroyed(ServletContextEvent arg0) {

@Override
public void contextInitialized(ServletContextEvent arg0) {
    // TODO Auto-generated method stub
    try {
        init();
    } catch (ServletException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
package src.googlemap;

//import trends.GetTrendingTopics;
//import twitter4j.TwitterException;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.google.gson.Gson;
import src.db.DbConnector;
import twitter4j.Trends;
import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.conf.ConfigurationBuilder;

@WebServlet("/GetMarkers")
public class GetMarkers extends HttpServlet {

    /**
     * 
     */
    private static final long serialVersionUID = 1L;
    private Connection con;
    private Statement st;
    Timestamp c;

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //response.setContentType("application/json");
        response.setContentType("text/html");
        try {
            java.util.Date date = new java.util.Date();
            con = DbConnector.getConnection();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
st = con.createStatement();
String query = "select * from trends ORDER BY date DESC, place limit 0, 20";
PreparedStatement ps = con.prepareStatement(query);
ResultSet rs = ps.executeQuery();
while(rs.next()){
    c = rs.getTimestamp(3);
}

Timestamp a = c;

Timestamp b = new Timestamp(date.getTime());
long diff = (b.getTime() - a.getTime()) / 60000;
if(diff > 10){
    ConfigurationBuilder cb = new ConfigurationBuilder();
    cb.setDebugEnabled(true)
    .setOAuthConsumerKey("sDbmBHO34Mef29VgskZe9H")
    .setOAuthConsumerSecret("7EClYiuFCNUKetLiaS3JsBuJFiFw3FI0pdR4dIH7jW1swTlEs")
    .setOAuthAccessToken("127786579-3va06nZ15Snvij6WESQcnGbQGRAPQio7dI8gn")
    .setOAuthAccessTokenSecret("UPe9eIEDxYcD6rerhJ5cVUhjTkkbz8SghqNnxK7yrDnvh");
    TwitterFactory tf = new TwitterFactory(cb.build());
    // ResponseList<Location> locations;
    // locations = twitter.getAvailableTrends();
    Integer idTrendLocation = 1;
    Integer idTrendLocation1 = 2459115;
    PrintWriter out = response.getWriter();
    response.getWriter().write("<h3><b>World Wide</b></h3>\n    for (int i = 0; i < 10 ; i++) {
        st.executeUpdate("insert into `trends` (place, trend, date) VALUES ('"+idTrendLocation+"', '"+trends.getTrends()[i].getName()+"', '"+new Timestamp(date.getTime())+"')");
        out.println("<a href=""+#"" onclick=""clickfunc(this)"">"+trends.getTrends()[i].getName() +"</a><br />")
    }
    System.out.println(trends.getTrends()[i].getName());
Trends trends1 = twitter.getPlaceTrends(idTrendLocation1);
for (int i = 0; i < 10 ; i++) {
    st.executeUpdate("insert into `trends` (place, trend, date) VALUE ('"+idTrendLocation1+"', '"+trends1.getTrends()[i].getName()+"', '"+new Timestamp(date.getTime())+"')");
    out.println("<a href="#" onclick="clickfunc(this)">"+trends1.getTrends()[i].getName()+"</a><br />");
    System.out.println(trends1.getTrends()[i].getName());
}
else {
    String queryx = "select * from trends ORDER BY date DESC, place limit 0,20";
    PreparedStatement psx = con.prepareStatement(queryx);
    ResultSet rsx = psx.executeQuery();
    int y =0;
    int x =0;
    while(rsx.next()){
        int i = Integer.parseInt(rsx.getString(1));
        if( i == 1){
            PrintWriter out = response.getWriter();
            if(y == 0){
                response.getWriter().write("<h3><b>World Wide</b></h3>);
                y++;
            }
            out.println("<a href="#" onclick="clickfunc(this)">"+rsx.getString(2)+"</a><br />");
        }
        else {
            PrintWriter out = response.getWriter();
            if(x==0){
                response.getWriter().write("<h3><b>New York</b></h3>");
                x++;
            }
        }
    }
}
out.println("<a href="#" onclick="clickfunc(this)"+"">"+rsx.getString(2)+"</a><br />"};
}
}

} catch (TwitterException | SQLException te) {
    te.printStackTrace();
    System.out.println("Failed to get trends: " +
    te.getMessage());
}

} public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    String filter = request.getParameter("filter");
    String sen = request.getParameter("sent");
    con = DbConnector.getConnection();
    try {
        String query = "select * from tweets where tweet_text like "
        + filter + "%" and sentiment like "" + sent + "%" ORDER BY created_at DESC limit 0,7000";
        System.out.println(query);
        PreparedStatement ps = con.prepareStatement(query);
        ResultSet rs = ps.executeQuery();
        Map<String, Map<String, String>> map_all_tweets = new HashMap<String, Map<String, String>>();
        while(rs.next()){
            Map<String, String> map_single_tweet = new
        HashMap<String, String>();
            map_single_tweet.put("tweet_id", rs.getString(3));
            map_single_tweet.put("screen_name", rs.getString(2));
            map_single_tweet.put("geo_lat", rs.getString(5));
            map_single_tweet.put("geo_long", rs.getString(6));
            map_single_tweet.put("tweet_text", rs.getString(4));
            map_single_tweet.put("profile_image_url",
            rs.getString(7));
            map_single_tweet.put("sentiment", rs.getString(9));
            map_all_tweets.put(rs.getString(3),
        map_single_tweet);
        } response.getWriter().write(new
Gson().toJson(map_all_tweets));
        con.close();
}
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

/**
 * Initialization of the servlet. <br>
 * @throws ServletException if an error occurs
 */
public void init() throws ServletException {
    // Put your code here
}
}
package src.Sentiment;

import org.json.*;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpURLConnectionExample {
    private final static String USER_AGENT = "Mozilla/5.0";
    // HTTP POST request
    public static String sendPost(String tweet) throws Exception {
        String url = "http://sentiment.vivekn.com/api/text/";
        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        //add request header
        con.setRequestMethod("POST");
        con.setRequestProperty("User-Agent", USER_AGENT);
        con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");
        String urlParameters = "txt="+tweet;

        // Send post request
        con.setDoOutput(true);
        DataOutputStream wr = new DataOutputStream(con.getOutputStream());
        wr.writeBytes(urlParameters);
        wr.flush();
        wr.close();

        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
        return response.toString();
    }
}
wr.writeBytes(urlParameters);
wr.flush();
wr.close();
int responseCode = con.getResponseCode();
System.out.println("\nSending 'POST' request to URL : " + url);
System.out.println("Post parameters : " + urlParameters);
System.out.println("Response Code : " + responseCode);
BufferedReader in = new BufferedReader(
        new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuffer response = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
in.close();
//print result
JSONObject jobj = new JSONObject(response.toString());
String sentiment = jobj.getJSONObject("result").getString("sentiment");
System.out.println(sentiment);
return sentiment;
}
References


10. https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline.


22. Twitter Data Analytics, Shamanth Kumar, Fred Morstatter, Huan Liu, pp. 35-40 August 19, 2013.


35. https://dev.twitter.com/streaming/overview

36. https://dev.twitter.com/streaming/public


38. https://developers.google.com/maps/documentation/javascript/

