Social Data Mining: Collection and Analysis of Political Posts

by

Ahmed Chaari

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

©Ahmed Chaari, 2016

Abstract

Social Networks become an important part of people's lives. A large number of individuals contribute to their contents. All aspects of everyday life – from work to politics, and further from entertainment to personal events – are reflected in posts generated on a daily basis. Those posts are perceived as a source of information that becomes a target and subject of analysis and research. This thesis addresses an issue of collecting and analyzing tweet posts. A comprehensive study of available programming tools suitable for collecting posts from multiple social networks has been conducted. Based on the results of that investigation, we have designed and developed a methodology for collecting tweets. Further, we have constructed a Social Data Mining Platform. The platform, implemented using Elasticsearch (open source full-text search engine), provides a number of approaches and algorithms for analysis of tweets. The analysis goes beyond processing of hast-tags and includes matching processes that involve the whole context of tweets. We have selected political tweets as the subject of our study. We have demonstrated how the proposed platform can be used to gain better understanding of political issues and opinions of a populace.

Preface

This thesis is submitted in fulfillment of the requirements for the degree of Master of Sciencein Software Engineering and Intelligent Systems. This thesis is an original work by Ahmed Chaari. No part of this thesis has been previously published.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Marek Reformat for his continuous support of my Master Program, for his motivation, constructive criticism, patience and engagement.

Last but not least I would also like to thank my parents for their endless love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Related Work

1.1 Introduction

In recent years, social networks (SNs) have facilitated connections between people by offering them an easy to use technology to share text, images, and videos, as well as the ability to chat. SNs have evolved very rapidly. They allow for keeping connection between friends, following most recent news and sharing peopole's thoughts. Those are just a few key examples that lead SNs to become extremely successful. The most known example of SN are Facebook, Twitter and Linkedin.

The success of SNs has attracted millions of active users on a daily basis which resulted in a huge amount of data created every second. Currently each SN is building its own social environment by offering developers APIs (Application Programming Interface) to develop applications that are connected their platform. Developers can create apps that access the users' data, update and manage their profile. The easy access to APIs has drawn the attention of researchers to analyze and interpret this data.

Social data mining is used in many areas like marketing, business intelligence, health monitoring, crises management, financial analysis and opinion mining. The data of SN offers an easy and continuous access that covers a wide range of people compared to data collected by surveys and polls.

1.2 Thesis goals and perspective

We set three main goal in this thesis. First create a generic SN data collector that can collect a huge data set very quickly and efficiently from multiple data Social networks .

Second, create a Social Data Mining Platform (SDMP) that is cloud ready, scalable and extensible.Our (SDMP) is an attempt to improve current social data analysis by taking a different approach that take inconsideration the text context in analyzing social data.
Third use the Data Collector and the SDMP for analysis of social data from Politics.We analyze the US Elections of 2012 by mining the social data related to the event.In our work, we want to investigate how a certain tweet from a regular user will relate to a each party. To achieve this goal we collect data sets containing tweets related to each party. That data is used to determine a unique signature representing each party. Based on those party signatures we classify users' tweets. This classification process is performed by calculating a score that reflect the relevance of the users' tweets to each party by taking in consideration hash-tags, noun phases and context form the users' tweets and the party tweets .Final results could be interpreted as a number of indicators that help make a more accurate and a guided prediction on which party wins

an election.

The proposed and developed SDMP can be generalized to utilize any social data. It can be applied to any SN posts to determine their relevance to certain entities. Such entities could be: topics, events, individuals, or organizations.

1.3 Related Work

Several papers haven been published that promote twitter as data source for data analysis in different areas. We can mention few examples like the paper of Jain and Kumar [14] where they Monitor Influenza-A (H1N1) epidemic spread in India in 2015. The paper of Gerber [10] introduced a system to predict crime in USA city of Chicago, Illinois. Another paper from Yang, Mo, and Liu [35] tried to predict the stock movement based on analyzing the sentiment of tweets from the users that form the financial community.

The first time that Twitter brought attention to the importance of social network in politics and in elections was in 2008. At that time the USA presidential candidate Barack Obama from the Democratic party started using twitter to promote his campaign. Twitter users from the Democratic parties participated actively in this campaign by sharing Obama tweets and using hash-tags to advertise the party plans after the elections and to attack the Republican party candidate John McCain. This strategy turn out to be very effective and was one of the key strengths of the Democrats which led to wining the 2008 elections.

After the 2008 USA elections, researchers gained more interest in using Twitter in politics. The first attempt of using Twitter for election prediction was shown in the paper Tumasjan et al. [32]. The authors tried to predict the outcome of the 2009 German federal Election. Other papers like Jungherr, Jurgens, and Schoen [15] criticized his paper and explained why the Pirates Party won the election that year instead of the Christian Democratic Union (CDU) as [32] predicted and pointed out several issues with their analysis like: the identification of the parties; and the date range for collection of tweets.

Other studies have showed more promising results, In study of the 2010 Swedish election campaign by Larsson and Moe [18]. The author did a statistical analysis of the data from Twitter using hash-tags and users mentioned in tweets and how they relate to each party using graphs. This study defined a simple guideline for future studies to have a better understudying of twitter. Further, in another paper form Ceron et al. [5] the study measure the popularity of politicians in Italy in 2011 and predict the voting intention for the second round of the presidential election that happened in France in 2012.The authors used HK method Hopkins and King [13] that performs a supervised Sentiment Analysis.They claimed that were able to get a Mean Absolute Error equal to 2.38 % for the french election which is very good prediction.Finally a paper from Burnap et al. [4] presents a sentiment based study of the UK 2015 elections, they pointed out an important issue with the current that other studies failed to mention, the issue of matching a tweet a party, in their data set only 19.4% of the tweets containing the term “Greens” were actually about the Green Party.A problem that we intend to address in this thesis. They used the sum of only used positive tweets to predict the vote share and seat share.In both studies of Ceron et al. [5] and Burnap et al. [4] the elimination of negative tweets from the results was not justified in their work , negative sentiment is an information that we intend to keep in our analysis because even if it's true that negative sentiment for a specific party will not give an indication on which party they have a positive sentiment to, we want to use that information to eliminate votes and to see the public reaction toward a certain party.

Chapter 2

Social Networks Data Collection

2.1 Introduction

Building a data set using social network (SN) data is a challenging task. It requires the use of a API(Application Programming Interface) or Library that will grant you access to the SN data, and manages the user's profile and his social activity. A property API is usually provided by the official company to allow developers to build apps. In the last years, major SN companies realized that offering an API right from the SN lunch will allow for quick development of apps. This will lead to build a whole ecosystem around SN and eventually will boost a number of users and generate more revenue.

The APIs are usually provided in multiple programming languages such as javascript, php, java, python. In most cases API are built as warpers of the SN official REST REpresentational State Transfer) APIs.REST APIs receive the most recent updates first and later the other APIs or Software development kits (SDKs). Generally, when API is not supported in a specific programming language, the open source community could also use the REST API to create a new API.

To create a data set, we need to obtain access to the API. This step is frequently achieved by filling a web form to request API keys or tokens. In most cases those keys are generated with respect to the *OAuth* [21] open specification. The keys allow developers to gain access and communicate with a given SN. In the next step, multiple scripts are created to query SN for data, download it and store it in a database for future use.

However, this process of creating a data set is quite difficult. Some of the most common obstacles are: APIs for the same SN in different programming language are not always synchronized in implementing all functions at the same time; documentation is quite often poor or incomplete; in some programming languages APIs are only available is open source format with low community support.

Each SN has its own APIs and data models. Therefore, it is very important and beneficial to construct a uniform social data mining platform (SDMP) in order to access several SNs. It would allow for collecting data multiple SNs and store it in a similar way without a need of major changes and updates in the code associated with each SN.
The idea of constructing such SDMP comes from identification of multiple similarities between data models for a number of SNs. For example:

- Personal SN: Facebook, Google+, VK Twitter
- Professional SN: Linkedin and Viadeo

- Location based SN: Yelp, Google Places, Foursquare and Tripadvisor
- Video based SN: YouTube, Vimeo and dailymotion.

Therefore, the goal is to find an approach/method that allows a generic access to different types of SN. This could be achieved by one of the following solutions:

- Finding a ready to use API that will cover all or most of the famous SNs with full implementation of all major functions/operations;
- building an API with an abstraction layer on the top of REST APIs;
- Finding APIs that implement access to different SNs in similar way, and use them with minimum modifications to access each SN.

In the next sections we will present some of those solutions that we tried, together with our feedback as well as the final solution that has been adopted to build our system.

2.2 Generic APIs

2.2.1 Introduction

The use of Generic API will allow us to do minimal code changes every time we add a new social network. It will ultimately use same functions for similar requests with few differences that reflect the specificity of each SN.

2.2.2 Open Source APIs vs. Proprietary APIs

Using a proprietary API offers some key advantages like the quality of code, the complete documentation and support, bug fixes and the availability of continuous and rapid updates. On the other hand, quite often proprietary license forces some limitations on how to obtain data, and how to perform specific tasks. All of that are obstacles for developers.

In some cases, the use of open source APIs is more suitable. The source code is accessible and customizable. There is also an access to a huge and active community support. In many cases, such support is considered better than the official support which is usually associated with a longer response time and does not offer alternatives or hacks for doing some specific tasks. However, some open source projects can stop offering support and updates. This happens if the project is maintained by few developers or if the interest in the project is lost.

2.2.3 Opensocial

The first solution that we looked into was *Opensocial* [24] . It started as an initiative solution launched by Google and MySapce in 2008. It offered an open source, multiple-system SN platform. It was introduced as an open source alternative to Facebook. The platform was based on html, javascript and Google Gadgets with the promise to provide interoperable applications among SNs that use the Opensocial API. *Apache Shindig* [2] is the reference implementation build with respect to the Opensocial specification that offers out of the box features to integrate Opensocial as a container.

We found out that a few SN companies like MySpace, Linkedin and SalesForce adopted the Opensocial for building their SNs. Un. As of December 2014, the Opensocial project stopped and moved to support the W3C Social Activity specification. In 2015 Apache Shindig has been retired.

2.2.4 OPENi

OPENi [23] is an open source project created by the European Union to develop a framework for integrating all major cloud services like SN. The goal was to build a centralized user cloud profile called Cloudlets. It's supposed to enable users to manage their personal information for each app and limit the disclosure of data that the app need to use to access multiple social networks and create cloud applications.

The OpenI framework is a very promising project. However, our tests showed that the first published releases are unstable. They lacked some key features which were under development and updates. Additionally, a number of individuals working on the project was still limited when compared to a number of developers contributing to the development of other APIs.It's still did not attract any contribution from the open source community.

2.2.5 Spring Social

Spring Social [30] is an open source framework for SNs written in Java. It was supported by Springsource – the company behind the Open Source Spring Frameworks Suit that included such components as: Spring core, Spring web service, Spring security, Spring data persistence, Spring mvc, Spring android, and so on. The suite that is widely used in Java programming.

Spring Social offers an easy integration with the other frameworks of Spring. However, the project did not offer any abstraction layer for a generic framework, but implemented an access to each SN as separate library on the top of Spring Social Framework core.

Spring Social has a good documentation, a growing community and contains an access to new SNs. It contains multiple features and supported major SNs like Google+, Facebook, Twitter and Linkedin from its first release.
Using Spring Social we implemented three social data collectors for Facebook, Twitter and Linkedin. Our experimentation has showed that development time was relatively longer than other Python based SN APIs. Therefore, we continued looking for a faster and easier alternative.

2.2.6 Mule ESB

Apache Software Foundation [3] is an open source Java based Enterprise Service Bus (ESB), as well as an application integration (AI) platform. It is built on open source standards and it implements in its design the best practices and recommendations for a middle-ware framework. The ESB is developer friendly. It uses a high level programming language based on XML. It also has a domain-specific language (DSL) called Mule Expression Language. Mule ESB is cloud ready, offers an API management tool and connectors for many Software as a Service (SaaS) applications.

The connectors offered by Mule ESB include connectors for Facebook, Twitter and Linkedin. For social connectors, the ESB warps a well-known open source java API in the background and delivers a straightforward configuration and implementation of all major functions.

Using Mule ESB, we implemented three specialized data collectors for Facebook, Twitter and Linkedin. Our tests with Mule ESB and Spring Social conducted on the same SN showed that there are some difficulties and challenges related to collecting social data. Those concerns will be detailed in the next section.

Chapter 3

Collection of Social Network Data

3.1 Limits of Twitter's API

The initial utilization of the RESTful *Twitter API* [33] has indicated a number of unfortunate limitations regarding the ability to download data. The free access is restricted with the following limits:

TABLE 3.1: Twitter API Rate Limits

Method	No of requests over a 15-min window (app auth)	Page Limit	History
GET statuses/ user_timeline	300	200 status	maximum of 3200 statuses per user
GET search/tweets	450	200 status	7 days and maximum of 3200 statuses

N.B. : These rates are subject to changes and they need to be checked regularly.

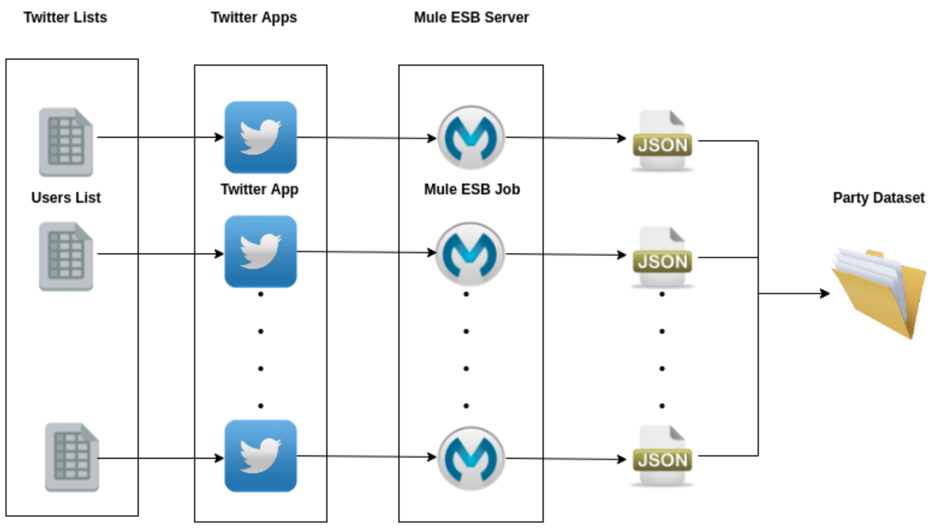
3.2 Implemented solutions to mitigate API Limits

In order to overcome the API's limitations we have looked at multiple options that would allow as to exceed the imposed limits. We have implemented a number of solutions.

3.2.1 Twitter User Timeline Collector

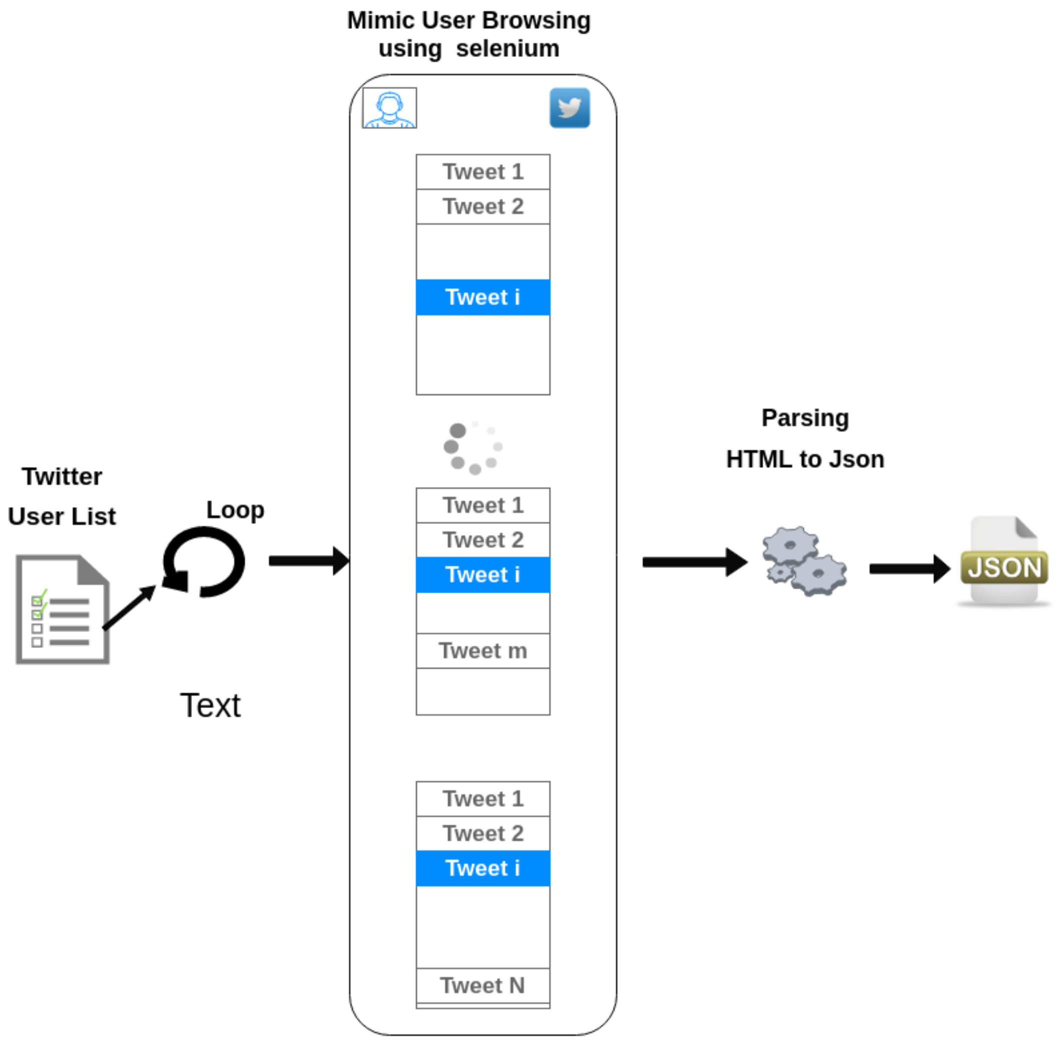
For the method *statuses/user_timeline*, we create multiple app auth (a maximum of 8 apps can be created) which give us an access to 8 different api keys on the Twitter developers website. In that case, we can run 8 processes in parallel to extract tweets. Each process is able to make 300 requests, and has to wait for 15 minutes time before it can start working again.

FIGURE 3.1: Twitter User Data timeline Collector



Additionally, for Twitter users that had more than 3200 tweets, we had to develop a web crawler, written in Python, that mimics the user's behavior in order to read all tweets using a Web Browser Automation tool called *Selenium* [29] before the date of the last tweet that we collected using the API. All collected tweets are saved in HTML format and parsed to JSON.

FIGURE 3.2: Twitter Web crawler



3.2.2 Twitter Search Collector

With the Twitter Search API, we can query for tweets that were tweeted in the 15 minutes period. Our solution was to create a twitter search collector that periodically collects and stores tweets using 8 different API keys and requests only tweets that have an ID higher than the last tweets we collected with the previous/last usage of the collector.

3.2.3 Tweet Status

The twitter API sends data in JSON format. Below, we show a sample of the tweet's status (only useful filed are shown):

Tweet Json Struture

```
{
      "text": "Teeet Text",
      "id": "tweet ID",
      "createdAt": "tweet timestamp",
      "retweetCount": "number of times the tweet was shared",
      "hashtagEntities": [{
            "text": "hashtag 1"
      }, {
            "text": "hashtag n"
      }],
      "userMentionEntities": [{
            "name": "User 1 Name mentioned in this tweet ",
            "screenName": "User 1 screenName"
      }, {
            "name": "User m Name mentioned in this tweet ",
            "screenName": "User m screenName"
      }],
      "user": {
            "followersCount": "number of users that Follow this user",
            "friendsCount": "number of the user's friends ",
            "biggerProfileImageURL": "User Profile Image URL ",
            "id": "user id",
            "description": "description of the user of this account",
            "location": "location of this user",
            "screenName": "a string that identify a user account",
            "lang": "tweet language",
            "statusesCount": "total number of status for this account",
            "timeZone": "user time Zoone in Text",
            "name": "Account user name",
            "createdAt": "user account creation timestamp",
            "utcOffset": "Timestamp utcOffset"
      }

}
```

3.3 Creating Data Sets

Our goal has been to create a data set related to politics that focus on election taken place in a countries (USA). That data set should contain two major parts: tweets from the party members participating in the election; and tweets from accounts of ’regular’ users, all data about the time of election.

3.3.1 Parties Tweets

The first part of the data set can be easily obtained with a list of Twitter accounts of members of a parliament and most important members of parties. Twitter has a feature called *twitter list* where you can create a collection of Twitter accounts for people to follow. Almost all parties share lists of party members, parliament members or party related accounts. Such lists enable to promote the party’s ideas, and make it easy to follow news related to the party. Also, some websites offer such lists for individuals to follow. In order to create our own lists for each party we merge all accounts that appear in those party lists in one list. Such created list will be used to collect tweets.

3.3.2 Regular Users Tweets

In the second part of the data set, we use the Twitter search API to search for tweets with specific words or hash-tags related to the election. However, this can only be done in real time before and during an election campaign. Such a campaign takes few weeks in Canada, and up to few months in US. So, to collect data we should wait for next election to happen. To overcome this problem, we have decided to use data set from the past elections, and we were able to find a big data set (170 millions status) of tweets by [20] collected in a three months period leading up to the 2012 US elections that was held on November 6th, 2012. The data has made available by data science consulting company called Kingmolnar. The following search terms have been used to create this data set:

TABLE 3.2: US 2012 Election Data Set Search Terms

Abortion	Democrat	MidWest	Republican
Afghanistan	DINO	Mitt	Rich
AfricanAmerican	economy	Morehouse	RINO
Ambassador	Embassy	Moveon	Romney
AnneRomney	energy	MarriageEquality	Romneycare
AUC	gas	NAACP	Senate
Bengali	HBCU	Nationaldebt	South
BirthCertificate	Heathcare	Obama	Spelman
Birther	HHS	Obamacare	SwingState
Black	Hurricane	Pakistan	Taxes
BlackVoter	Iran	PaulRyan	TeaParty
Bluestates	Israel	Poll	Undecided
CAU	Jerusalem	Poor	VoterID
ChristopherStephens	Liberal	POTUS	VotingRights
Congress	Libya	Purplestates	West
Conservative	Media	Prolive	Yemen
CivilRights	MichelleObama	Parenthood	Youth
Debt	MiddleClass	Redstates	Youthvot

The 2012 elections were a very large event in the US history. They consist of the following multiple elections:

- the 57th presidential election;
- Senate elections;
- House of Representatives elections.

Social Data Mining is a Big Data problem, which requires powerful hardware to run. However, we only had access to a standard PC. Also, we wanted to be sure that the tweets we have are related to the US election – it was important because some search terms could be associated with tweets that are not related to the election. That required us to filter all tweets from the downloaded data set using the hash-tag *#Election2012*. That hash-tag was the official hash-tag used by all parties to indicate relation to the 2012 Election and all tweets created before November the 6th of 2012.

Chapter 4

Social Data Mining Platform

4.1 Introduction

4.2 Data Set Enrichment

Before we start processing and analyzing our data set, we need to enhance our raw data by adding two extra attributes: Noun Phrases List and sentiment polarity. The values of these attributes are calculated based on the tweets' text using Textblob Python natural language processing (NLP) library.

4.2.1 Noun phrases

A noun phrases (NP) is a phrase that contain a noun such as a person, a place, an object. Often, a noun phrase is just a noun or a pronoun. We use the list of extracted noun phrases in a similar way as hash-tags. This is particularly useful when we have a tweet that contains a few hash-tags.

For example, in the sentence: *In 2012 election, Obama became the eleventh President and third Democrat to win a majority vote more than once.*

The noun phrases are :

- Election
- Obama
- President
- Democrat

4.2.2 Sentiment Analysis

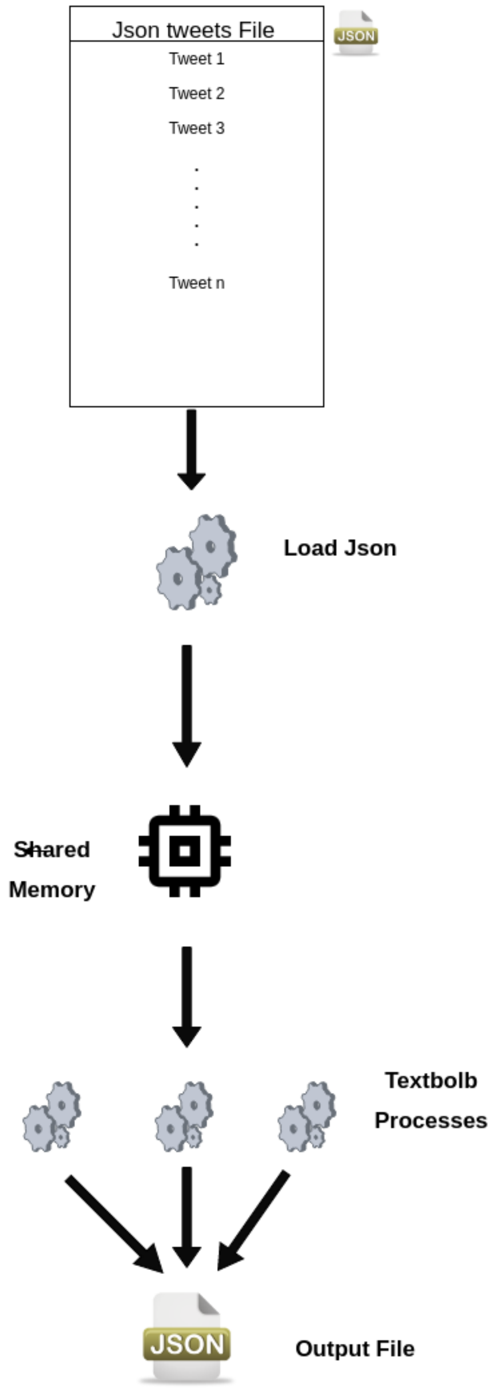
In order to calculate the sentiment polarity, where polarity is a float number within the range [-1.0, 1.0] we use a *TextBlob* [31] Naive Bayes Classifier. The classifier is trained on the movie review corpus, and is imported from the leading Python NLP library called *Python Natural Language Toolkit(NLTK)* [26].

4.2.3 Code Tuning and Parallel Processing

Our test showed that processing a tweet using Texbolb to extract the noun phrases and to calculate the sentiment polarity on a PC with a two-core CPU and a 3 GHz clock requires about 2 seconds. To reduce that time we started optimizing our code using multiple approaches to speed-up the process of loading JSON files. We used a fast python package called *python cjson library* [25] that creates a warper on the top of a C based JSON library. We compared a JSON loading time obtained with CJSON with solutions using several other packages (jsonlib, simplejson, ujson and yajl), but we only gained a few microseconds.

The next step was to improve our code using Python parallel processing. After implementing the multiprocessing algorithms, our code was running 4 processes simultaneously and with this change we managed to decrease the processing time for one tweet to an average of 1 second. Figure 4.1 illustrate this process.

FIGURE 4.1: Textbolb parallel processing



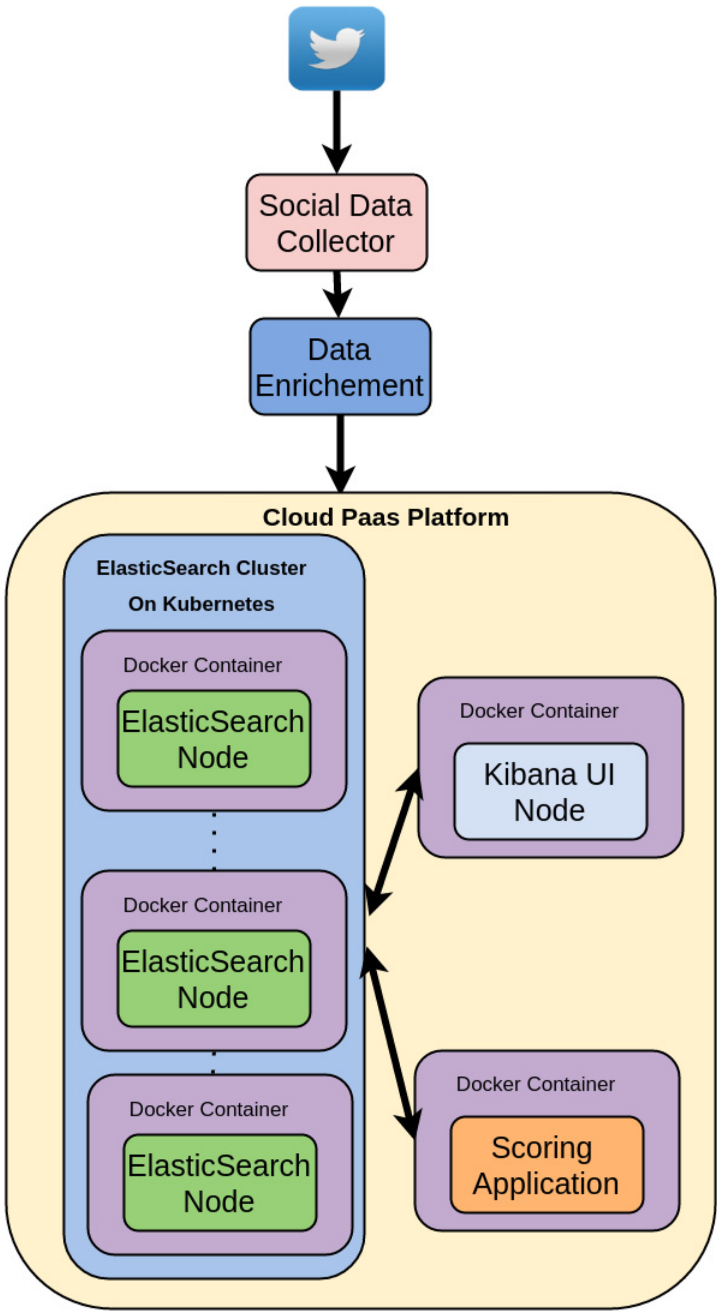
To improve our processing time even more we experienced with using *Opencl* [22] library in python for Graphics processing unit (GPU) programming , the problem was that opencl uses C99 version of C language for the parallel portion of code , and because all of our code is in python , we needed to rewrite some libraries like textbolb in c99 , which is a very hard task.

4.3 Platform Architecture

The process of building our social data mining platform (SDMP) required careful selection of every technical aspect of the system in order to satisfy a number of quality attributes. The platform should be: robust, easily scalable, cloud ready, and sustainable.

The platform has been tested on the PaaS cloud provider [11]. We have used Elasticsearch [**Elasticsearch**] as a document based database and a real time full text search engine that scores tweets based on specific queries. Elasticsearch runs a cluster of multiple nodes managed using Kubernetes [17], each node is a container running separately. The system UI is based on Kibana UI running inside a *Docker* [6] container and we use several Python scripts running inside a container to query Elasticsearch and updates its indices.

FIGURE 4.2: Social Data Mining Platform



4.3.1 Kubernetes based Platform as a Service (PaaS)

Kubernetes [17] is an open source project from Google to create and manage a cluster of containerized applications.

The system has been tested on the cloud Paas Google Container Engine that is built on the top Kubernetes and on local server running Mincube version of Kubernetes.

Kubernetes allows an easy deployment and management of docker containers in pods. A pod is similar to a host machine where containers share Linux namespaces, local network and docker volumes.

4.3.2 Docker

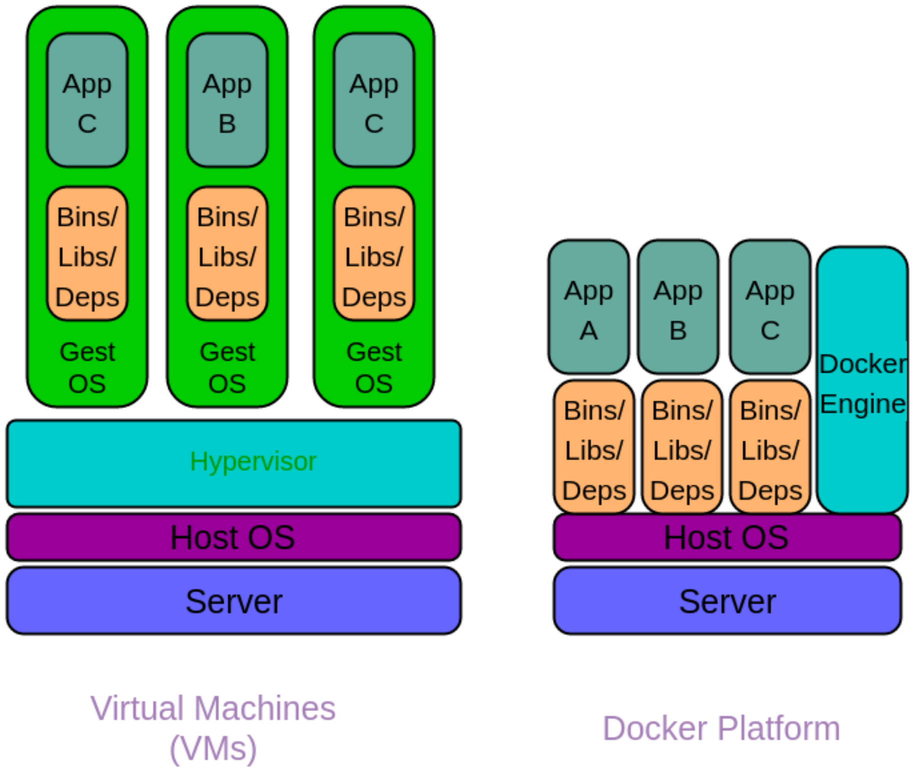
Docker is an open source platform create by DotCloud Inc. (Now Docker Inc.). It allows to create a distributed application encapsulated in an isolated environment called containers. The containers are based on Linux Containers (LXC) and are created on the base of a docker image. A docker image is a a collection of binary files assembled based on a specification in a dockerfile (kind of a configuration file). A single container contains dependencies to run an application inside it. The application can communicate with the host system through the network and can use its file system.

Docker evolved quickly and gained a lot of support from many open source communities and from large IT companies. It became a platform for manginging containers that can work together and communicate between each other. Docker is used to build a strong back-end system for many organizations.

Docker also eliminates hassles of system administrators and developers of applications that behave differently when deployed on different development and production environments. Since Docker 'uses' containers that are built using the same image containing the same libraries and packages it runs anywhere and guarantees the same outputs. It enables an approach *write once, run anywhere (WORA)*.

The benefit of using Docker containers is the fact that they use the host Linux kernel to perform all tasks, no matter what version of Linux OS (Red Hat, Ubuntu ,CentOS) is run on a host machine. This results in a small size of container images. Each container is running as sperate process on the host machine. In contrast, Virtual Machines (VMs) are

FIGURE 4.3: Virtual Machines VS Docker Platform



a full virtual environment that contains a layer of virtualization for the targeted environment – hardware and resources.

Docker images are usually created using a dockerfile which is the source code used to built an image. Typically, this file contains steps how to install the application that suppose to run, as well as all dependencies of packages and libraries. It also contains instructions how the container file system communicates with the host, which ports are exposed to the host machine, and how to load external configuration files. Every line of this file is compiled and it results in a separate layer on top of the layer created during the execution of the previous line. Eventually, the whole image is created. The user pushes the image to a private or public repository. Later it can be used to create containers.

Docker containers have become widely used and many open source projects are currently offering a stable docker image to make sure that their application will perform as expected. Such companies like Amazon, Google and Microsoft offers IaaS (infrastructure as a service) and PaaS (Platform as a service) cloud services supporting docker containers. Currently, docker is the best solution to make sure any application will work without any issues on the cloud environment.

4.3.3 Elasticsearch Introduction

ElasticSearch [7] is an open source text search engine. It includes NoSQL document based database and a text mining tool built on top of Apache Lucene. Elasticsearch hides the complexity of using Lucene and uses it internally as the core of its platform. Elasticsearch offers a RESTful API to interact with a client as one server. Both requests and responses use JSON. It stores data as a document oriented database that represents one of the types of Key-value databases. It means that we do not use tables and rows as in conventional relational databases but we store the whole document. The document can also be seen as an object in object-oriented programming. By default, such objects have no schema what makes them more compact. Additionally, we do not need to have a value for each variable like in a relational database.

Each document inserted in Elasticsearch is automatically indexed in JSON format. We call this process indexing. Documents are stored in a way similar to a relational database. They can have a type. Types can be compared to a table in relational databases and basically they are filters that help organize documents. To query Elasticsearch, we need to create a query using a Domain Specific Language (DSL) based on Json, and use the API get method.

In our data set, all tweets are indexed in one index and using seven types: six types for the parties (each one contains tweets from the same party), and one more type to store regular users tweets.

In the next sections related to Elasticsearch, we will explain the main internals components of Elasticsearch. The technical details are based on the *Elasticsearch Documentation* [9], and the book by Gormley and Tong [12]. We also provide details how these features are used in the process of building our social data mining platform (SDMP).

4.3.4 Elasticsearch Filtering

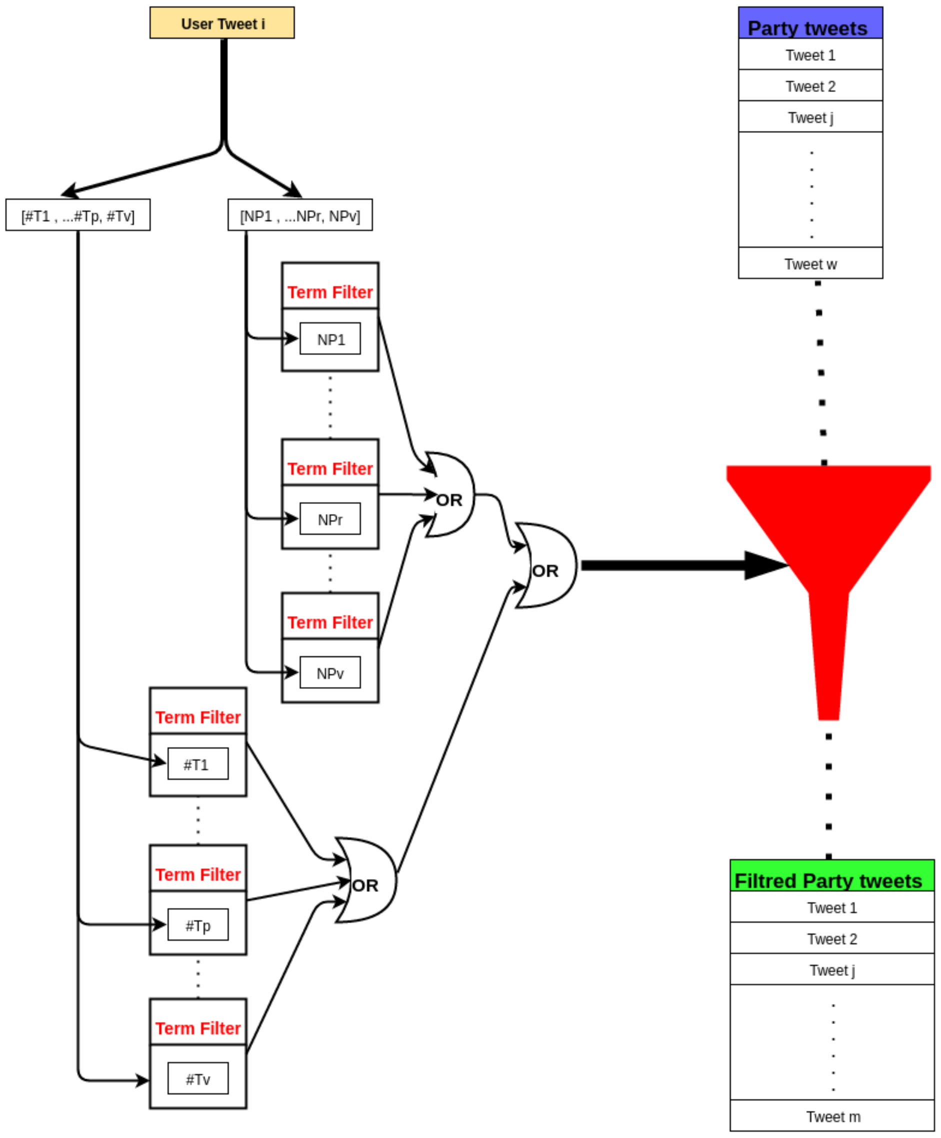
Elasticsearch Filters works as a true of false classifier. The user creates a filter in the Query DSL which contains a collection of conditions and criteria applied on an index or a type which species if a document should have to pass a filter or not.

Documents that successfully passed the filter will be later used in the query body of the search request. For example, the filter query built to find tweets with a sentiment polarity grater than 0.2 that were created between 01/04/2011 and 06/11/2012 in our US elections data set is shown below:

```
"filter": {

           "bool": { "must": [
            {"range": {
             "createdAt": {
               "gte": "01/04/2011",
               "lte": "06/11/2012",
               "format": "dd/MM/yyyy"
             }
            }}
            ,
            {"missing" : { "field" : "parties_agg" }}
            ,
            {
                     "range": {
                       "tweet_sentiment_polarity": {
                         "gte": 0.2
                       }
                     }
            }

           ]}
          }

}
```

In Figure 4.4: Party Score Query Filtering Algorithm, we represented the filtering process for scoring user tweets. Filters are dynamically created based on the User tweets to eliminate irrelevant party tweets. The

FIGURE 4.4: Party Score Query Filtering Algorithm



elimination is done based on basic term filters for the hash-tags and the noun phrases. A document is considered relevant if it contains at least one hash-tag or one noun phrase from the user original tweet. Those generated filters will be added in the whole Scoring Query. The scoring part of the query assigns higher scores to the most relevant filtered documents. The scoring process is composed of several steps which will be discussed in details in the next sections.

4.3.5 Elasticsearch Analyzer

An analyzer in Elasticsearch is basically an NLP tokenizer which split a string into terms or tokens, and remove stop words.

Several analyzer are available in Elasticsearch by default. We have picked the English analyzer because it is the most convenient one for our data set US 2012 Elections. This analyzer was used for all tweets treated as string fields. The configuration of the analyzer has been set up using Elasticsearch mapping. This process will be discussed later in a separate section.

The tokonezation is done at the time of indexing (inserting) a tweet. This helps Elasticsearch to be very quick at data search and scoring.

All major languages have ready to user analyzers. So for another data set we can pick the analyzer that supports the language used in the data. We can also customize analyzer by modifying one of the basic ones.

The English analyzer has a list of stop word for the English language, as well as an English term stemmer.

For example, the listing below represents an implementation of the English analyzer as a customized analyzer built using basic tokenizers, from *Elasticsearch Documentation* [9] :

```
{
 "settings": {
  "analysis": {
   "filter": {
    "english_stop": {
     "type": "stop",
     "stopwords": "_english_"
    },
    "english_keywords": {
     "type": "keyword_marker",
     "keywords": []
    },
    "english_stemmer": {
     "type": "stemmer",
     "language": "english"
    },
    "english_possessive_stemmer": {
     "type": "stemmer",
     "language": "possessive_english"
    }
   },
   "analyzer": {
    "english": {
     "tokenizer": "standard",
     "filter": [
      "english_possessive_stemmer",
      "lowercase",
      "english_stop",
      "english_keywords",
      "english_stemmer"
     ]
    }
   }
  }
  }
}
```

Note. Data from Elasticsearch Documentation

URL: www.elastic.co/guide/en/elasticsearch/reference/current/analysis-lang-analyzer.html

To extend the default analyzer, we use two other Character Filters: the HTML Strip Char Filter returns a clean string without any HTML elements; and the Mapping Char Filter which can be used to create a mapping table between emoticons (emojis) to their meanings in English.

4.3.6 Elasticsearch Term Vectors

Term vector in Elasticsearch contains statistics about the output of the Analyzer. Two term metrics could be extracted per term or per document (tweet) over the entire index:

- ttf: total term frequency (it is a number of occurrences of a given term in index)
- doc_freq : document frequency (a number of tweets that contain a given term in index).

To extract term vectors we use the Multi-termvectors API which permit querying for term vectors in multiple tweets using a list of tweets ids. This list makes the processing time shorter by reducing the number of requests over the network and we will use Terms filtering to only retrieve the most relevant term vectors using minimum term frequency of 100.

In the future sections we will also be using other performance boosting features like bulk methods.

The term vector statistics allows us to determine a unique party signature (or fingerprint), which is later used by Elasticsearch to score user tweets.

4.3.7 Multi Search

To query Elasticsearch we will use the Multi Search API instead of the regular Search API. The idea is the same as in the bulk indexing. In our algorithm, we have a large number of queries to trigger, and this creates a lot of overhead. In order to mitigate this issue we query the index in bulk, which means we have to prepare each query and its metadata. To automate this process, we have created three Dynamic Query Builders in Python that use a query template and add relevant query data for each of them: One Dynamic Query Builder for scoring regular users tweets against each party tweets; and two Dynamic Query Builder to aggregate these scores in two steps.

4.3.8 Elasticsearch Similarities

A similarity in Elasticsearch or Apache Lucene is an algorithm to calculate a weight of each term which will be used in matching formula to calculate a relevance score based on all terms weights.

Lucene has several similarity algorithms. We will compare the results using two similarities calculated on all fields for the party and the regular users tweets. We will describe the algorithm in next sections.

Term Frequency–Inverse Document Frequency (TF_IDF) Similarity

We are summarizing the TF/IDF similarity used by Elasticseach and described in details in *Elasticsearch Documentation* [9]

The term weight in TF/IDF is calculated based on three values calculated and stored during indexing (inserting) of tweets:

Term frequency (TF) : the prevalence of a term t in a document d (tweet), Lucene uses the square root of the number of occurrences of the t in d

$$tf(t,d) = \sqrt{frequency}$$

The Inverse document frequency (IDF) Lucene uses the logarithm of the number of filtered documents (tweets), divided by the number of filtered documents that contain the term.

The more the term is popular, the lower its IDF value is. IDF is used to minimize the weight of common terms.

$$idf(t) = 1 + \log \frac{number\ of\ filtered\ documents}{(number\ of\ documents\ contain\ the\ term + 1)}$$

Field-length norm (norm) To assign higher weight to a shorter field in order to increase its chances to be more relevant.

$$norm(d) = \frac{1}{\sqrt{number\ of\ Terms}}$$

The field-length norm is calculated using the inverse square root of the number of terms in the field.

Term Weight(TF_IDF) : The final term weight is the product of all those three previous values:

$$weight(t,d) = tf(t,d) \cdot idf(t) \cdot norm(d)$$

Okapi BM25 (Best Matching 25) Similarity :
Elasticsearch uses the Robertson Stephen et al. [27] version of Okapi BM25. In this section we present a basic explanation of the Okapi BM25 similarity based on Robertson [28] paper and Weber [34] presentation.

The BM25 Formula can be divided in three parts :

$$BM25(d) = \sum_{t \in q, f_{t,d} > 0} \boxed{\log(1 + (\frac{N - df_t + 0.5}{df_t + 0.5}))} \cdot \boxed{\frac{f_{td}}{f_{td} + k}} \cdot \frac{1}{\boxed{(1 - b + b\frac{1(d)}{avgdl})}} \tag{4.1}$$

- Part 1 represents the IDF factor : Term popularity
- Part 2 represents the Saturation Curve factor :limit the effect of TF on the score
- Part 3 represents the Length Weighing factor: change the influence of document length

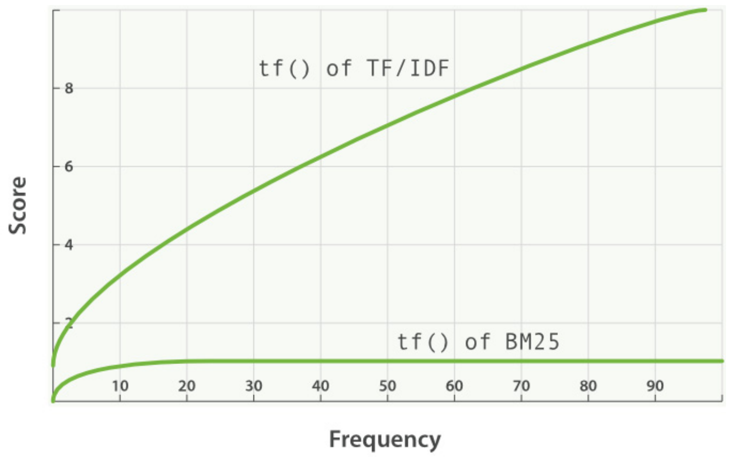
where :

- df_t=number of document that contain the term
- N=Number of Documents
- f_{td} = frequency of term in document

- k = saturation parameter
- b= length parameter
- l(d) = number of tokens in document
- avgdl = average document length in corpus

BM25 formula is quite complex when compared to TF/IDF. It was created using multiple mathematical approximations. The most important advantage of BM25 compared to TF/IDF is how it behaves when a term appears many times. In TF, the term that appears 20 times has more effect on relevance than terms that appears 5 times, while in BM25 the term that appears 10 and 20 times are treated the same way. This is called a "nonlinear term-frequency saturation" and is illustrated in Figure 4.5

FIGURE 4.5: limit influence of tf on the score



Note. Figure from *Elasticsearch Documentation* [9]
link:
https://www.elastic.co/guide/en/elasticsearch/guide/current/pluggable-similarites.html

Another advantage of BM25 over TF/IDF, is that it is tunable by adjusting two parameters: k and b:

- k: controls a term-frequency saturation; we use the default value of 1.2 which is the recommended for most types of documents;
- b: controls an effect of field-length normalization; a value of 0.0 disables normalization, while a value of 1.0 normalizes fully; we use a value of 0.75 which is the recommended for most types of documents.

4.3.9 Elasticsearch Mapping

Mapping in Elasticsearch means adding index configuration metadata per field. In our data set the mapping for our data set includes the choice of: the similarity algorithm, the analyzer algorithm; and the type and format of each field.

Setting the mapping of an index the right way is very important because unlike relational databases where we can easily make changes regarding type and structure of the data, in Elasticsearch a lot of processing, like tokenizing and term weight calculations, happens at the time of indexing a document (tweet in our case) and depends on the algorithm that we choose. We have to re-index all documents for most of changes we make in the mapping. Unfortunately if the mapping contains some parts of the old mapping, Elasticsearch does not use values that are calculated before. Therefore, it is recommended to start with tests on small data sets and to keep changing the mapping until desired results fit our needs.

4.3.10 Elasticsearch Bulk Insert and Update

Using the regular indexing API to insert and update document is not very practical. Especially, if we need to insert or update a big data set that should be re-indexed time to time. The two main issues are high indexing time and high network bandwidth usage. To boost this process we use the Bulk API to push data to Elasticsearch in chunks. Each data chunk is a file that contains data from the data set of tweets together with indexing metadata like index name, type, and document id.
We have created Linux shell scripts and Python scripts to automate this process and dynamically create and add metadata. Using the Bulk feature we have been able to minimizing the time of indexing and updating the whole data set in Elasticsearch on a sigle node from six hours to one hour.

4.3.11 Elasticsearch Query Matching

To classify user tweets we start with scoring tweets from each party separately. It is done dynamically via building a query based on the text of the user's tweet, hash-tags and noun phrases. Results of scoring calculations perform on each party are aggregated and compared to find the party with the highest aggregated score.

After filtering, we use a full text matching called "Match Query". It allows us to perform an exact-value term search in the tweet's text. We also add a query boost for hash-tags matching to double the importance of hash-tags when compared to the text of a tweet. This decision was made after a data set exploration phase.

In Figure 4.5, we illustrate how the query is build based on a tweet from a regular user. This query takes the output of the filter as input and Use Lucene Practical Scoring Function with the specified similarity algorithm and terms weights to compute the score for each tweet that passed the filter. In the next sections we provide more details regarding the used algorithm.

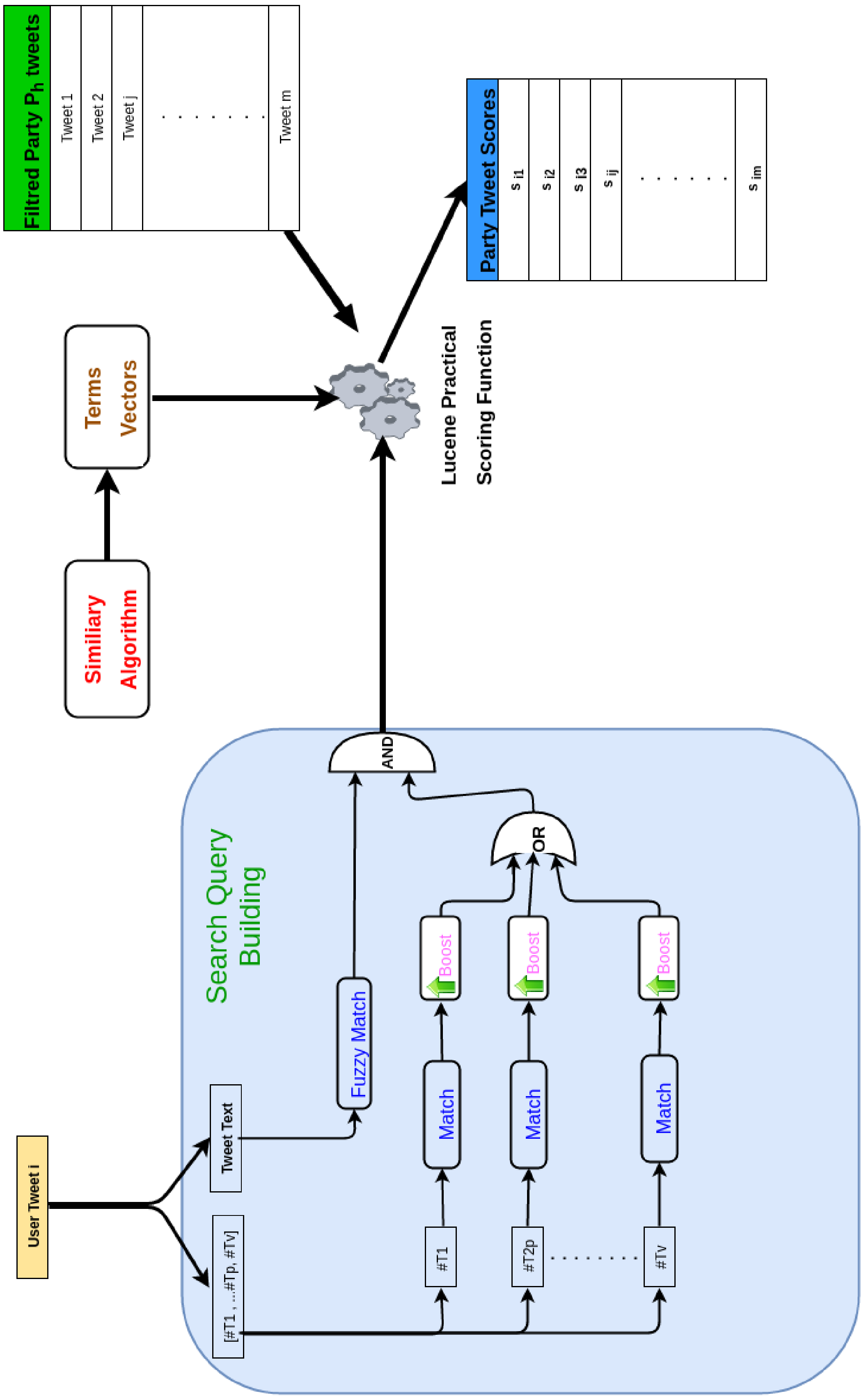


FIGURE 4.6: Tweet Scoring Algorithm

4.3.12 Elasticsearch Fuzzy Query

In order to ensure a bit more flexible text matching, we use an Elasticsearch option called 'fuzzy query'. This helps in identifying terms that are mistyped or contain repeated characters.

The fuzzy matching query uses the Damerau-Levenshtein edit distance (Damerau 1964; Levenshtein 1966) for a text field. Based on the fuzzy match query parameters, Elasticsearch determines all possible terms by doing a number of character modification operations: deletion; insertion; substitution and transposition; called edits. This operation is controlled by: the limit of max expansions; and the edit distance.

We use a recommended value of 50 for the default max_expansions, and the AUTO edit distance. The AUTO edit distance means that depending of the length of a term the following behavior is expected:

- $[0; 2]$ no edits, the query must match without fuzziness;
- $[3; 5]$ one edit is allowed;
- > 5 two edits are allowed.

```
"match": {
  "text": {
  "max_expansions" : 50
  "fuzziness": "AUTO",
  "query": "tweet text"
}
```

4.3.13 Lucene Practical Scoring Function

This section explains how the final relevance score is calculated in Elasticsearch. Our explanation will be based on the book by Gormley and Tong [12] and the *Apache Lucene Documentation* [1].

Vector Space Model Vector Space Model is actually a vector that contains weights terms. Both document and query will be represented as multi-term vectors. The weights in the Vector Space Model are calculated using the similarity algorithm. The problem of comparison was solved by comparing the angle between multidimensional vectors via computing Cosine similarity $\cos\theta$. For two non-zero query vector $\vec{q}$ and document $\vec{d}$:

$$\vec{q}\cdot\vec{d} = \|\vec{q}\|\|\vec{d}\|\cos\theta$$

cosine similarity equation :

$$\cos\theta = \frac{\vec{q}\cdot\vec{d}}{\|\vec{q}\|\|\vec{d}\|}$$

Lucene Practical Scoring Function formula Lucene Practical Scoring Function formula is a combination of the Boolean model and the vector space model.We based our explination on the *Elasticsearch Documentation* [9] and the *Lucene Practical Scoring Function* [19].The Boolean model simply filters the documents. An approximation of the vector space model is used and its refining is done by introducing multiple factors.

$$\begin{aligned} score(q,d) &= coordFactor(q,d) \cdot queryBoost(q) \\ &\cdot \frac{\vec{q}\cdot\vec{d}}{\|\vec{q}\|} \cdot docLenNorm(d) \cdot docBoost(d) \end{aligned} \tag{4.2}$$

Many terms in this formula are calculated during indexing at the time a document (tweet in our case) is added to Elasticsearch:

- Query-boost for the query is from the query q.
- Document length norm doc-Len-Norm(d) and document boost doc-Boost(d) values can be computed in advance at indexing time and their multiplication is saved as norm(t in d) where t means term.

For TF/IDF similarity, the rearranged final formula is as follows:

$$\begin{aligned} score(q,d) &= coord(q,d) \cdot queryNorm(q) \\ &\cdot \sum tf(t\ in\ d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d) \end{aligned} \tag{4.3}$$

where:

- tf(t in d): the Term's Frequency, calculated based on the formula introduced in the TF/IDF Similarity Section;
- idf(t): the Inverse Document Frequency, calculated based on the formula introduced in the TF/IDF Similarity Section;
- coord(q,d): Query Coordination, a higher score is given to documents that have more query terms

$$coord(q,d) = \frac{overlap}{maxOverlap}$$

where :

overlap: a number of query terms that exist in the document

maxOverlap: a total number of terms in the query

- queryNorm(q): Query Normalization Factor, Lucene attempt to normalize query scores to be able to compare it. However, this comparison is not very accurate if scores form different queries are compared

$$queryNorm = 1/\sqrt{sumOfSquaredWeights}$$

For a Boolean Query (OR, AND) the Sum of Squared Weights is :

$$sumOfSquaredWeights = q.getBoost()^2 \cdot \sum_{t\ in\ q} (idf(t) \cdot t.getBoost())^2$$

- t.getBoost() is a search time boost of term t in the query q as specified in the query text

- norm(t,d) encapsulates Field boost f.boost() and length factor length-Norm

$$norm(t,d) = lengthNorm \cdot \prod_{field\ f\ in\ d\ named\ as\ t} f.boost()$$

4.3.14 Elasticsearch Kibana UI

To have a complete social data mining platform (SDMP), we have added a user interface layer. We utilize a plugin for Elasticsearch called Kibana ElasticSearch [8] that allows to visualize data. Kibana helps in creating quick and interactive visualization. It is very useful in data exploration phase to query the index in real time and view the results quickly. After we have implemented our algorithm which include filtering, querying and aggregation, the final results are stored in the form of tables so Kibana has been not used. For more complex visualization a full customization is possible. Another good alternative is to use the web application Shiny and create scripts in R language to query and plot data. Here, the advantage is an access to all powerful R packages.

Chapter 5

Data Set , Analysis Methodology & Results and Discussion

5.1 Data Set Statistics

Table 5.1 shows the basic statistics related to data sets containing tweets from parties involved in US elections 2012. The process of collecting those tweets was described in Chapter 3, Section 3. As it can be seen some parties had more tweets than others, this problem will be addressed in the aggregation phase of our algorithm to minimize 'the size effect' on our analysis.

TABLE 5.1: Data Set Tweets Count for Parties

Party	**Republican**	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Number of Tweets	95193	95731	62618	8625	2128	13202
Number of Twitter Accounts	560	361	43	175	16	63

5.2 User Tweet Matching and Scoring Methodology

5.2.1 Users Data Filtering

First we filter the users' tweets to eliminate tweets that are neutrals, i.e., tweets with $-0.2 \leq SentimentPolarity \leq 0.2$. Those tweets are not very useful in our analysis since they do not carry the users' position toward a party. The output of this filter will be our new regular a users data set TU_F_N.

5.2.2 Party Matching

To match or classify a user tweet tu_i to a certain party, we calculate scores that represent the relation between the user tweet and each tweet from the party. We repeat this process for each party P_h where $h \in [1, 6]$, and $i \in [1, n]$ with n representing a total number of user's tweets. Before scoring tweets, we filter tweets from each party to eliminate ones that are not relevant to the current user tweet tu_i.

5.2.3 Parties Tweets Filtering

To filter parties tweets, we dynamically build a filter $F_i(tu_i)$. This filter eliminates irrelevant tweets from all tweets of a given party tp_he where $e \in [1, w]$ and w is a number of tweets for P_h. The output of this filter represents the filtered tweets of a party tp_hj where $j \in [1, m]$ and $m \leq w$. This process is described in Figure 4.4. We use the same filter F_i on all parties for each user tweet tu_i, so the number of tweets that pass the filter depends on the filter and party.

5.2.4 Filtered Users Tweets Statistics

Table 5.2 contains statistics about user tweets after all neutral tweets are removed. Our users tweets data set TU_F_N is a collection of positive tweets T_+ and negative tweets T_-.

TABLE 5.2: Sentiment based Filtered Users Tweets Statistics

Tweets Filter	All Users	Users T_+	Users T_-	Users T_N
Tweets Count	88011	20555	7663	59793

T_+ :Users tweet with $SentimentPolarity \geq 0.2$
T_- : Users tweet with $SentimentPolarity \leq -0.2$
T_N :Neutral Users tweet with $-0.2 \leq SentimentPolarity \leq 0.2$

The filtering of the data set TU_F_N processes can be presented in two stages. First, there is filtering of tweets that belong to each party based on the user's tweet. It is possilbe that the user's tweet would not have any relevent tweets from a given party. Second, the scores represtning maching of tweets of a given user to tweets from all parties could be zero. The score values depend on the applied similarity algorithm. Table 5.3 contains statics about tweets data set TU_F_N after filtering, and for different similarity algorithms and the sentiment polarity.

TABLE 5.3: Similarity and Sentiment based Filtered Users Tweets Statistics

Tweets Filter	Users TF_IDF T_+	Users BM25 T_+
Tweets Count	5058	3484

Tweets Filter	Users TF_IDF T_-	Users BM25 T_-
Tweets Count	2369	1369

T_+ :Users tweet with $SentimentPolarity \geq 0.2$
T_- : Users tweet with $SentimentPolarity \leq -0.2$

5.2.5 Scoring

The output of the filtering process is the input for the scoring. Query $Q_i()$ is applied. It uses a similarity algorithm to determine a score $score_{ij}$ between the user's tweet and each tweet from the party tweets tp_he where $i \in [1, n]$ and $j \in [1, m]$. Each query $Q_i()$ is built dynamically from fields in the user's tweet tu_i. This process is already explained in Section 4.3.11 and Figure 4.6.
It is very important to notice that the same query is applied to each party. This makes aggregated score for each party comparable per a user tweet.

5.2.6 Aggregation

To aggregate the scores for a given user's tweet t_i across all party tweets tp_hj, we calculate the sum of squares $Agg_{ij} = \sum_{i,j=1}^{n,m} score_{ij}^2$. Therefore, for each user's tweet we have a vector:

$$Agg_i = \begin{bmatrix} Agg_{i1} \\ \vdots \\ \vdots \\ Agg_{i6} \end{bmatrix}$$

The sum of squares provides us with a measure representing the magnitude of scores associated with each party. The sum of squares will boost tweets that have higher scores and minimize the effect of tweets with lower scores. This makes sense since the scores for the user's tweet against one parties will only have few tweets with high scores while the majority of scores will be close to zero. We do not want to eliminate tweets with low scores because they still carry information related to matching of tweets. It is the result of the filtering stage where all tweets that contain at least one hash-tag or noun phrase from the user tweet are selected. The sum of squares is also useful to make aggregated score results from parties that contain less tweets comparable to the other aggregated score results from parties with high number of tweets.

5.2.7 Party Matching and Classification

We assign M_ij for a user tweet t_i across all party tweets tp_hj, M_ij takes the value of $M_{ij} = Agg_{ij}$ if $Agg_{ij} = \max(Agg_1..Agg_6)$ and 0 otherwise.

5.2.8 Tweets Party Matching and Scoring Results

Finally, in order to interpret the results we split the users tweets to two sets: one set of positive tweets T^+; and one of negative tweets T^-. We analyze each set separately. We do not include neutral tweets in the final results because they do not provide any valuable information regarding sentiment of the user to the party.

- T_+: Users with tweets with $SentimentPolarity \geq 0.2$
- T_-: Users with tweets with $SentimentPolarity \leq -0.2$
- T_N: Users with tweets with $-0.2 \leq SentimentPolarity \leq 0.2$

In order to have a better understanding how users thought about parties and how popular they were, a number of indicators have been proposed:

- We calculate a score called $AvgofMaxScore$ that combine the sentiment and the aggregated scores M_{ij} of a party

  $$AvgOfMaxScore_h = Avg(M_{ij} \cdot sent_i)$$

  this score gives us a general indication of pepole's attitude toward a certain party and how strong it was.
- We calculate a rank of parties based on descending values of $AvgOfMaxScore_h$
- We calculate FrequencyOfMaxScore by counting a number of $M_{ij} \neq 0$ for party P_h, this score reflects the voting tendency because even if one likes or hates a party so much her vote still counts as one; here we assume that each tweet represent the user's opinion about a given party.

Results from the last three measures are in table 5.4 and table 5.5

- We calculate Frequency$T^{+} - T^{-}$ by subtracting FrequencyOfMaxScore for T^{-} from FrequencyOfMaxScore for T^{+}.

$$FrequencyT^{+} - T^{-} = FrequencyOfMaxScore_{T^{+}} - FrequencyOfMaxScore_{T^{-}}$$

Results of $FrequencyT^{+} - T^{-}$ are in table 5.6

TABLE 5.4: Positive Tweets Results

	Republican	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Avg of Max Score TF IDF T_{+}	4.182	8.286	2.459	0.148	0.222	0.464
Rank TF IDF T_{+}	2	1	3	6	5	4
Frequency of Max Score TF IDF T_{+}	1142	1373	875	123	1039	506
Avg of Max Score BM25 T_{+}	4185.915	9959.371	1692.346	61.792	16.587	267.25
Rank BM25 T_{+}	2	1	3	6	5	4
Frequency of Max Score BM25 T_{+}	1026	1576	567	48	41	226

TABLE 5.5: Negative Tweets Results

	Republican	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Avg of Max Score TF IDF T_-	-4.147	-19.449	-2.08	-0.102	-0.085	-1.228
Rank TF IDF T_-	2	1	3	5	6	4
Frequency of Max Score TF IDF T_-	547	740	443	66	237	336
Avg of Max Score BM25 T_-	-3541.749	-21712.28	-1135.709	27.002	-9.098	-804.541
Rank BM25 T_-	2	1	3	5	6	4
Frequency of Max Score BM25 T_-	378	651	194	21	9	116

TABLE 5.6: Frequency T_+ - T_-

	Republican	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Frequency TF IDF (T_+ - T_-)	479	836	124	-18	-196	-110
Frequency BM25 (T_+ - T_-)	648	925	373	27	32	110

5.3 Election Results

In table 5.7 we include USA 2012 election results from Wikipedia. These results allow us to compare our findings with the real election results.

TABLE 5.7: Actual Election Results Retrieved from Wikipedia

Presidential candidate	Party	Popular vote Percentage
Barack Obama(Incumbent)	Democratic	51.06
Mitt Romney	Republican	47.20
Gary Johnson	Libertarian	0.99
Jill Stein	Green	0.36
Virgil Goode	Constitution	0.09
Roseanne Barr	Peace and Freedom	0.05
Rocky Anderson	Justice	0.03
Tom Hoefling	America's	0.03
Other		0.17

Note. Data From Wikipedia
URL : https://en.wikipedia.org/wiki/United_States_presidential_election ,_2012

5.3.1 Algorithm validation

To validate our algorithm and convince ourselves that it works, we have created two data test sets. The first one has contained a subset of Tweets from the members of Democratic party, while the second one contained a subset of Tweets from the members of Republican party. Tweets in both data test sets have been used as the users' tweets. The results obtained from our algorithm provided expected outcome. The tweets of users – members of each party in reality – have been classified (matched) correctly to their parties.

Results of the validation process for the Democratic and Republican parties are presented in Table 5.8 and 5.9

TABLE 5.8: Party Classification Algorithm Validation for the **Democratic Party**

	Republican	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Frequency of Max Score / TF IDF	2	113	0	0	0	0

Test Data Set consist of 115 tweets (after filtering) form the Democratic Party

TABLE 5.9: Party Classification Algorithm Validation for the **Republican Party**

	Republican	**Democrats**	**Justice**	**Green**	**Socialism and Liberation**	**Libertarian**
Frequency of Max Score / TF IDF	94	5	0	0	0	0

Test Data Set consist of 99 tweets (after filtering) form the Republican Party

5.4 Term Vectors

Term Vector for each party represent the signature of that party. They give us a good idea of the most important topics and interests discussed in it. Terms vectors details for each party are included in Tables 5.10 to 5.15. We also added Word-Clouds of these vectors in Figures 5.1 to 5.6.

TABLE 5.10: Term Vectors of the Republican Party

Term	**ttf**	**doc_freq**
job	9173	8439
4job	6067	6057
bill	5488	5284
here	5465	5410
tax	4510	4072
obama	4294	4211
vote	4134	3901
hear	4050	3963
senat	3694	3547
american	3694	3621
budget	3634	3365
presid	3614	3551
us	3601	3446
discuss	3586	3571
pass	3259	3146
gop	3223	3167
tcot	3163	3163
debt	3154	2978
work	3023	2955
energi	3020	2735

ttf: total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq: document frequency (the number of tweets that include a term in in the party's index)

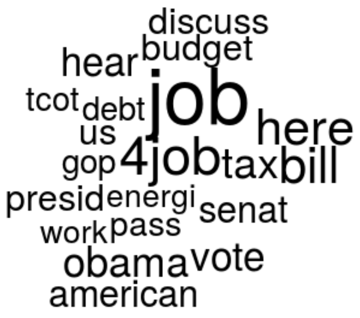


FIGURE 5.1: Word cloud of Term vector from the Republican party

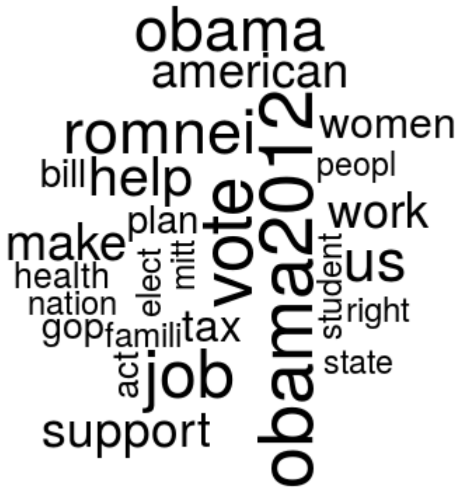


FIGURE 5.2: Word cloud of Term vector from the Democratic party

TABLE 5.11: Term Vectors of the Democratic Party

Term	**ttf**	**doc_freq**
barackobama	8544	8538
obama2012	6996	6980
vote	6567	5844
job	6124	5512
presid	5676	5529
romnei	5647	5503
obama	5540	5422
us	5244	4951
help	4793	4679
make	4295	4141
support	4156	4080
american	4039	3936
work	3987	3823
tax	3591	3176
women	3378	3071
bill	2987	2861
plan	2963	2874
gop	2838	2763
act	2658	2609
health	2564	2421

ttf :total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq : document frequency (the number of tweets that include a term in in the party's index)

TABLE 5.12: Term Vectors of the Justice Party

Term	**ttf**	**doc_freq**
occupi	5039	4769
peopl	2571	2438
occupywallstreet	2352	2331
protest	2346	2291
polic	2040	1961
just	1955	1920
arrest	1774	1677
vote	1687	1480
wiunion	1550	1550
tcot	1544	1544
teaparti	1448	1414
sai	1445	1388
support	1444	1397
call	1396	1351
march	1386	1331
obama	1381	1330
make	1366	1314
occupyseattl	1329	1276
street	1240	1210
occupyoakland	1232	1158

ttf :total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq : document frequency (the number of tweets that include a term in in the party's index)

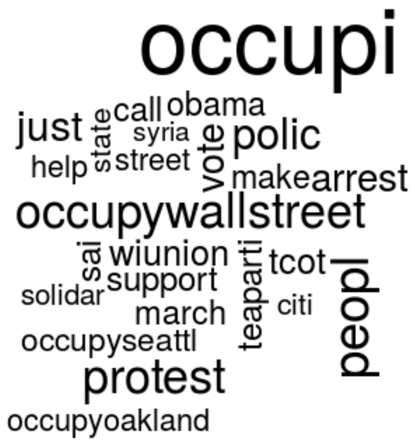


FIGURE 5.3: Word cloud of Term vector of the Justice party

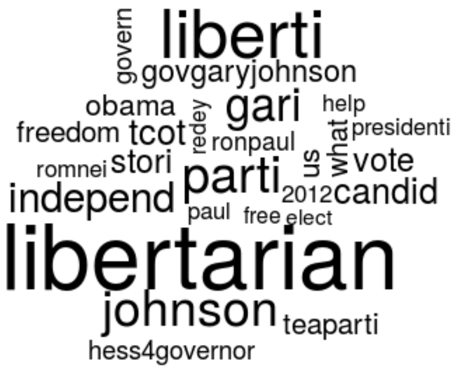


FIGURE 5.4: Word cloud of Term vector of the Libertarian party

TABLE 5.13: Term Vectors of the Libertarian Party

Term	**ttf**	**doc_freq**
libertarian	2158	2062
liberti	1576	1553
parti	1082	1042
johnson	1060	1007
gari	936	896
independ	825	823
tcot	660	659
candid	656	627
vote	588	500
stori	579	579
govgaryjohnson	524	521
obama	500	481
us	488	459
freedom	484	472
teaparti	472	471
what	454	423
hess4governor	397	397
govern	396	380
ronpaul	372	353
romnei	340	331

ttf :total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq : document frequency (the number of tweets that include a term in in the party's index)

TABLE 5.14: Term Vectors of the Socialism and Liberation Party

Term	**ttf**	**doc_freq**
votepsl	483	421
petalindsai	313	313
candid	233	232
campaign	229	224
lindsai	202	202
presidenti	200	200
election2012	197	197
yari_nobord	175	172
live	166	159
peopl	141	135
right	135	128
us	134	127
social	122	117
fight	117	113
osorio	98	98
elect	91	91
polic	89	85
mikeprysn	89	89
protest	86	85
romnei	340	331

ttf :total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq : document frequency (the number of tweets that include a term in in the party's index)

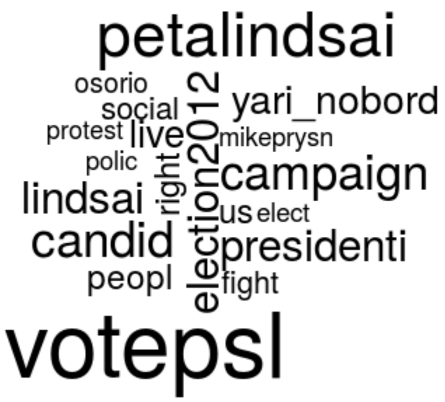


FIGURE 5.5: Word cloud of Term vector of the Socialismand Liberation party

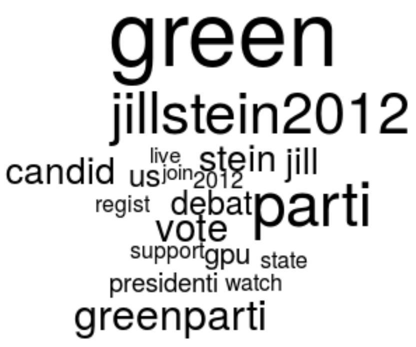


FIGURE 5.6: Word cloud of Term vector of the Green party

TABLE 5.15: Term Vectors of the Green Party

Term	**ttf**	**doc_freq**
green	2481	2224
parti	1597	1448
jillstein2012	1468	1402
greenparti	978	975
vote	859	719
stein	809	797
candid	806	789
jill	780	766
debat	677	627
us	619	576
gpu	539	523
presidenti	458	453
2012	360	356
state	355	339
watch	349	343
support	348	334
regist	347	325
live	321	311
join	313	310
romnei	340	331

ttf :total term frequency (the number of is the number of occurrences of a term in all tweets in the party's index)
doc_freq : document frequency (the number of tweets that include a term in in the party's index)

5.5 Term Context

From parties term vector we notice that for example the term obama appears in almost every party .In many other studies which only uses terms without it context and the measure of it's relevance, this will create many false positives. This proves that our methodology should be more accurate. To further investigate this issue and show an example of how we can create a context of term from each party , we used the term obama in each party to find tweets that contain this term and we calculated the frequency of hash-tags and noun-phrases that coexistence in the same tweet for all the tweets. And we used this results to create Word cloud of the context of the term obama for all parties.

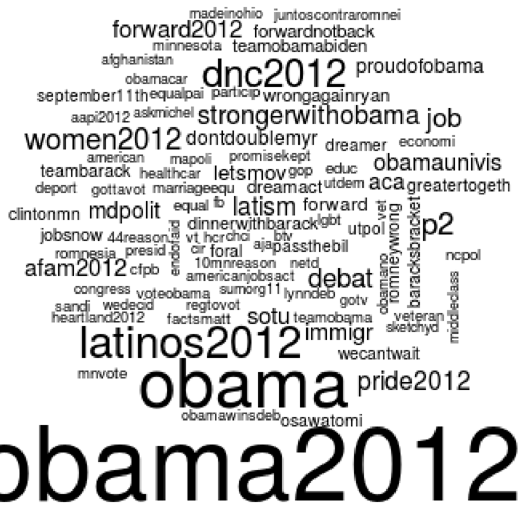


FIGURE 5.7: Word cloud from terms in hash-tags hash-tags from Democratic party that relates to the term Obama

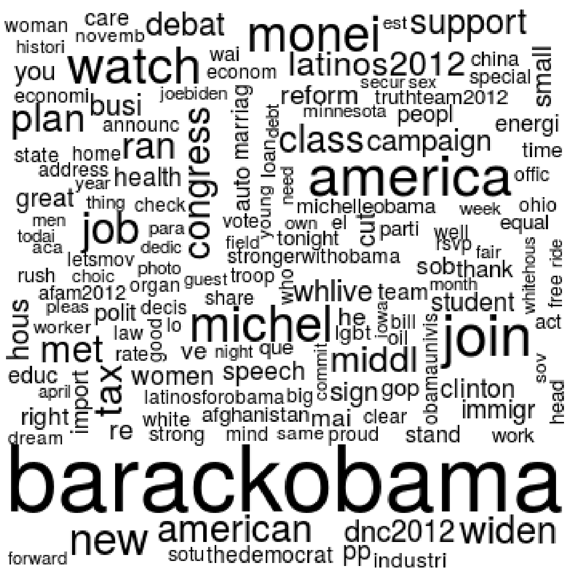


FIGURE 5.8: Word cloud from terms in noun phrases from Democratic party that relates to the term Obama

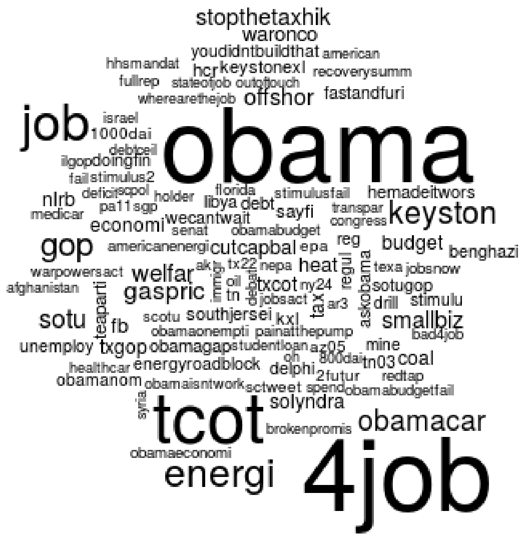


FIGURE 5.9: Word cloud from terms in hash-tags from Republican party that relates to the term Obama

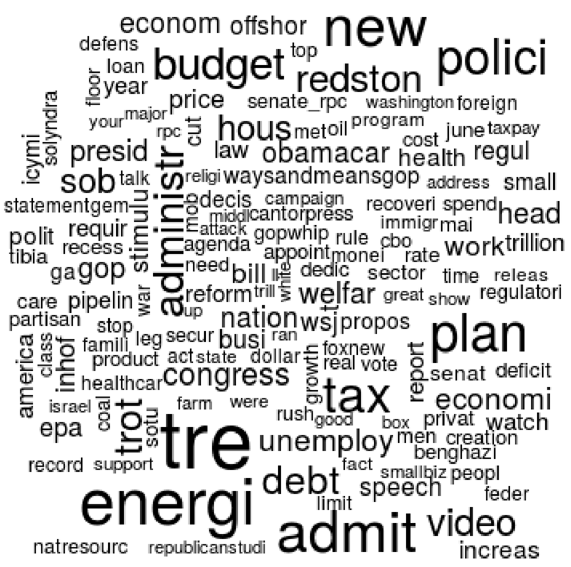


FIGURE 5.10: Word cloud from terms in noun phrases from Republican party that relates to the term Obama

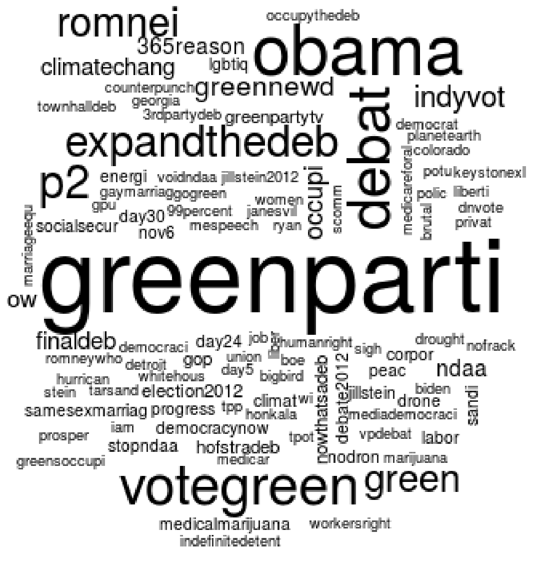


FIGURE 5.11: Word cloud from terms in hash-tags from Green party that relates to the term Obama

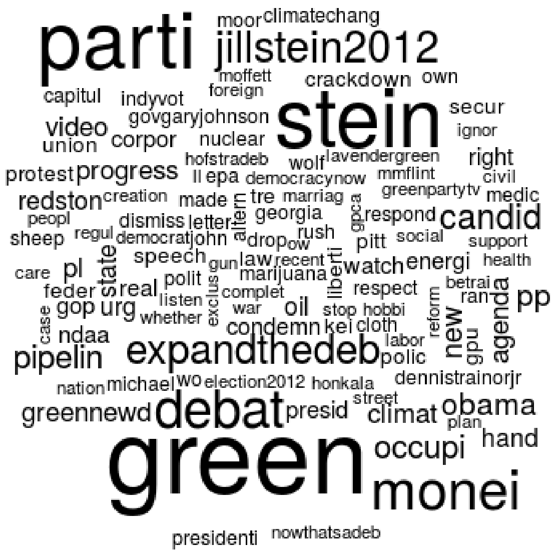


FIGURE 5.12: Word cloud from terms in noun phrases from Green party that relates to the term Obama

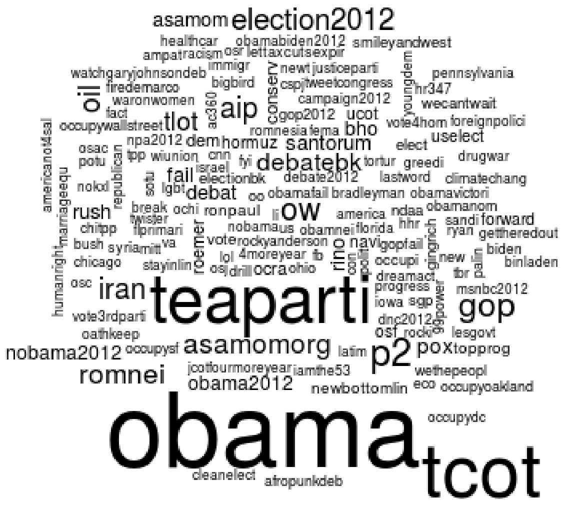


FIGURE 5.13: Word cloud from terms in hash-tags from Justice party that relates to the term Obama

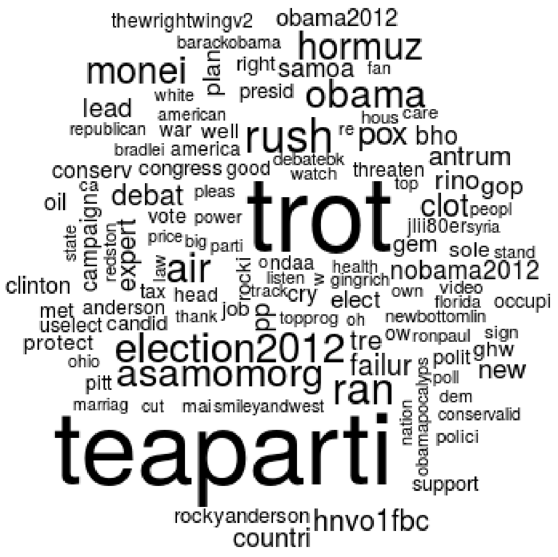


FIGURE 5.14: Word cloud from terms in noun phrases from Justice party that relates to the term Obama

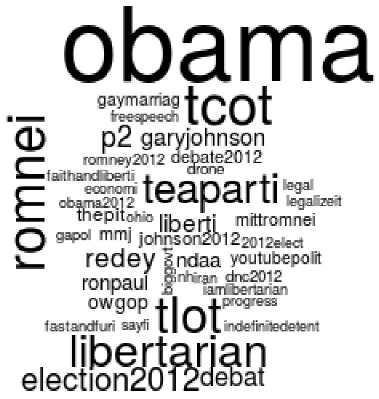


FIGURE 5.15: Word cloud from terms in hash-tags from Libertarian party that relates to the term Obama

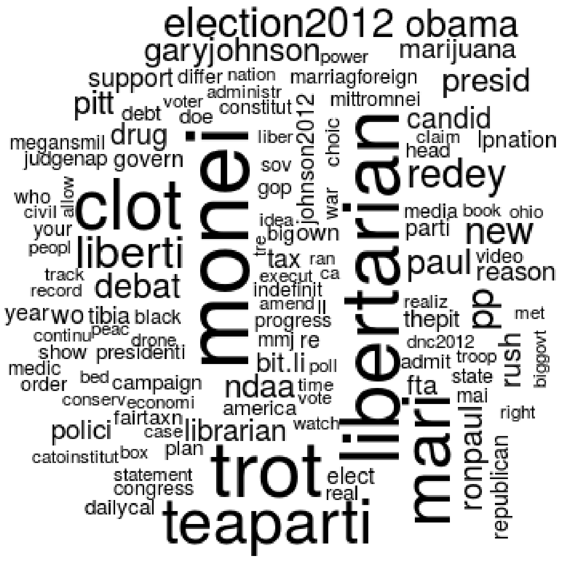


FIGURE 5.16: Word cloud from terms in noun phrases from Libertarian party that relates to the term Obama

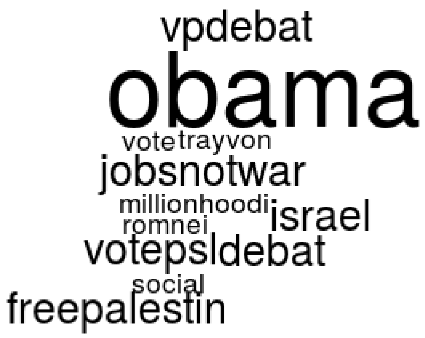


FIGURE 5.17: Word cloud from terms in hash-tags from Socialism and Liberation party that relates to the term Obama

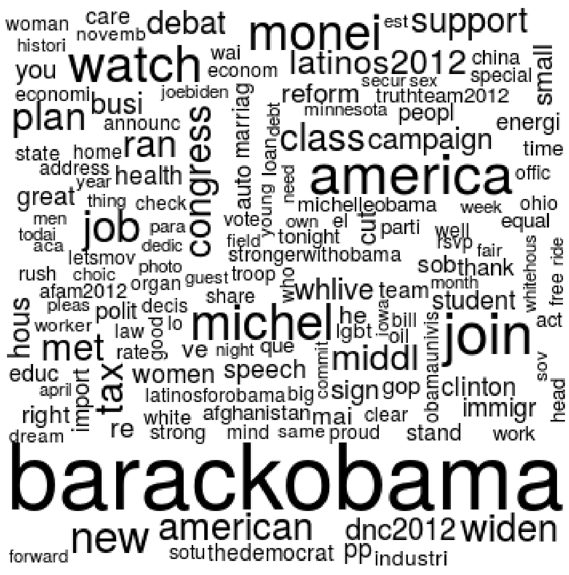


FIGURE 5.18: Word cloud from terms in noun phrases from Socialism and Liberation party that relates to the term Obama

5.6 Results Interpretations and Discussion

From these results, by using the $FrequencyT_{+} - T_{-}$ measure, we can easily detect that the Democratic party is most likely to win this election which was the case in the US elections 2012. The Republican party comes second with a small difference. More people felt positively towards the Democratic party.

Another interesting finding is that people felt negatively, were almost angry at the Democratic party. However, because each person can only make one vote, this does not affect negatively the Democratic party.

The Justice party was third in our results but it did not received much votes in the elections. To understand this we need to look at the term vector of this party, where we can easily spot that this party was strongly supporting the Popular movement 'Occupy Wall Street' which had support from Americans from different parties. Another important factor is that the two-party system in USA forces a large number of people to vote for the Democrats or Republicans to ensure that the other candidate will not be elected.

For The Green, Socialism and Liberation and Libertarian parties we can detect that results changes depending on the scoring algorithm TF/IDF and BM25. This happened because the aggregation score of each tweet was very low and this can easily be affected by the selection of similarity algorithm. TF/IDF is good because it is easy to compute but from examining the results and the data set content, we can can see that BM25 is more accurate. This confirms the decision to use BM25 in Elasticsearch and Lucene. BM25 is a default similarity algorithm in the newer version of Elasticsearch.

Chapter 6

Contributions, Conclusion and Future Work

6.1 Contributions

In this thesis we have shown that our three initial goals have been achieved. First, we created social data collectors that can be easily modified and extended to cover other social networks (SN). It is true that we were limited by privacy policies in some SNs and by the API rates in others, but this issue for industrial application can be solved via purchasing the appropriate perineum API access.

Second we implemented a Social Data Mining Platform (SDMP) based on Elasticsearch that can scale easily and can be extended to use with other SNs. SDMP is also cloud ready and very quick since parts of the processing happens in inserting time.

Third we used our platform to test it with analysis of tweets related to the US Elections 2012. We presented a new way of matching tweets to entities(organizations ,events ,peoples,communities ...) that takes in consideration many factors like hash-tags, noun phrases, text of tweets, and entity unique signature.

6.2 Conclusion

Building a social data mining platform and making sure that it is cloud ready and easily extensible was a challenging task. It required the use of several new technologies that had to be integrated to compose the desired system. Another big challenge was to work with a data set containing 'political' tweets. It is know that this topic is hard when includes sentiment analysis and social mining.

In our analysis, we obtained interesting results that we believe genuinely reflect the content of the data set. The proposed approach leads to better understanding of social data that uses common language which carry a certain ambiguity.

6.3 Limitation and Future Work

There are few limitation to our study. Firstly, a bigger data set of users' tweets will give a more accurate results. Secondly, adding emotion analysis like the open source project by Krcadinac et al. [16], would increase the depth of findings. Emotion detection in text is a relatively new field of research. It is related to sentiment analysis that started to bring attention of researchers. Using this technique will give us a more accurate understanding of people reaction to parties by knowing their emotions like joy, fear, sadness, anger.. . Finally adding twitter location data will allow us to have an idea about the popularity of each party in every Electoral district or seat. To add this feature to the analysis we need to build an address validation algorithm since we cannot use only the geolocation field in a tweet – this option is not enabled by the majority of users. We can combine geolocation field with the user location text field which in input manually to obtain an address. To achieve this the user's location needs to be normalized, mapped and scored depending on their relevance to a list of addresses taken as a reference.

From the current results we can find two possible future extension of this social data mining platform using a training data from the score results of our algorithm. The first one is development of a recommender system that is able to match individuals to the party that best fits their interests. The second possible extension it is to train a neural network to automatically classify tweets to parties.

In the presented work Twitter and political tweets were used as the example to test our algorithm. This work can easily modified to use data sets from other social networks related to other interest area or community. For example, we are currently considering modifications of the algorithm to detect how people react to the use of green technologies from tweets collected in different cities. We are also considering using our current analysis for the next US presidential elections of 2016.

Bibliography

[1] *Apache Lucene Documentation*. URL: https://lucene.apache.org/core/documentation.html.

[2] *Apache Shindig*. 2009. URL: https://shindig.apache.org/.

[3] Apache Software Foundation. *Mule ESB*. 2010. URL: https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb.

[4] Pete Burnap et al. "140 characters to victory?: Using Twitter to predict the UK 2015 General Election". In: *Electoral Studies* 41 (2016), pp. 230–233. ISSN: 02613794. DOI: 10.1016/j.electstud.2015.11.017. arXiv: 1505.01511. URL: http://dx.doi.org/10.1016/j.electstud.2015.11.017.

[5] a. Ceron et al. "Every tweet counts? How sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to Italy and France". In: *New Media & Society* 16.2 (2013), pp. 1–19. ISSN: 1461-4448. DOI: 10.1177/1461444813480466. URL: http://nms.sagepub.com/cgi/doi/10.1177/1461444813480466.

[6] *Docker*. 2010. URL: www.docker.com.

[7] ElasticSearch. *ElasticSearch*. Version 2.3. URL: https://www.elastic.co/products/elasticsearch.

[8] ElasticSearch. *Kibana*. Version 4. Aug. 20, 2015. URL: https://www.elastic.co/products/kibana.

[9] *Elasticsearch Documentation*. URL: https://www.elastic.co/guide/index.html.

[10] Matthew S Gerber. "Predicting crime using Twitter and kernel density estimation". In: *Decision Support Systems* 61 (2014), pp. 115–125.

[11] Google. *Google Cloud*. URL: https://cloud.google.com/.

[12] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. " O'Reilly Media, Inc.", 2015.

[13] Daniel J. Hopkins and Gary King. "A method of automated non-parametric content analysis for social science". In: *American Journal of Political Science* 54.1 (2010), pp. 229–247. ISSN: 00925853. DOI: 10.1111/j.1540-5907.2009.00428.x.

[14] Vinay Kumar Jain and Shishir Kumar. "An Effective Approach to Track Levels of Influenza-A (H1N1) Pandemic in India Using Twitter". In: *Procedia Computer Science* 70 (2015), pp. 801–807.

[15] A. Jungherr, P. Jurgens, and H. Schoen. "Why the Pirate Party Won the German Election of 2009 or The Trouble With Predictions: A Response to Tumasjan, A., Sprenger, T. O., Sander, P. G., & Welpe, I. M. "Predicting Elections With Twitter: What 140 Characters Reveal About Political Sentiment"". In: *Social Science Computer Review* 30.2 (2012), pp. 229–234. ISSN: 0894-4393. DOI: 10.1177/0894439311404119. URL: http://ssc.sagepub.

com/content/30/2/229$%5Cbackslash$nhttp://ssc.sagepub.com/cgi/doi/10.1177/0894439311404119.

[16] Uros Krcadinac et al. "Synesketch: An open source library for sentence-based emotion recognition". In: *IEEE Transactions on Affective Computing* 4.3 (2013), pp. 312–325.

[17] *Kubernetes*. 2014. URL: http://kubernetes.io/.

[18] a. O. Larsson and H. Moe. "Studying political microblogging: Twitter users in the 2010 Swedish election campaign". In: *New Media & Society* 14.5 (2012), pp. 729–747. ISSN: 1461-4448. DOI: 10.1177/1461444811422894.

[19] *Lucene Practical Scoring Function*. 2016. URL: https://lucene.apache.org/core/4_6_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html.

[20] Peter Molnar. "Twitter 2012 Presidential Election Data Set". In: (July 2015).

[21] *OAuth*. 2014. URL: https://oauth.net/.

[22] *Opencl*. 2016. URL: https://www.khronos.org/opencl/.

[23] *OPENi*. 2014. URL: http://www.openi-ict.eu/.

[24] *Opensocial*. 2008. URL: https://www.w3.org/blog/2014/12/opensocial-foundation-moves-standards-work-to-w3c-social-web-activity/.

[25] *python cjson library*. 2016. URL: https://pypi.python.org/pypi/python-cjson.

[26] *Python Natural Language Toolkit(NLTK)*. 2016. URL: http://www.nltk.org/.

[27] E Robertson Stephen et al. "Okapi at TREC-3". In: *Proceedings of the Third Text REtrieval Conference (TREC 1994). Gaithersburg, USA*. 1994.

[28] Stephen Robertson. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2010), pp. 333–389. DOI: 10.1561/1500000019.

[29] *Selenium*. 2016. URL: http://www.seleniumhq.org/.

[30] *Spring Social*. 2014. URL: http://projects.spring.io/spring-social/.

[31] *TextBlob*. 2016. URL: https://textblob.readthedocs.io/en/dev/.

[32] a. Tumasjan et al. "Election Forecasts With Twitter: How 140 Characters Reflect the Political Landscape". In: *Social Science Computer Review* 29.4 (2011), pp. 402–418. ISSN: 0894-4393. DOI: 10.1177/0894439310386557.

[33] *Twitter API*. 2016. URL: https://twitter.com/rest/public.

[34] Britta Weber. *Improved Text Scoring with BM25*. Mar. 2016.

[35] Steve Y Yang, Sheung Yin Kevin Mo, and Anqi Liu. "Twitter financial community sentiment and its predictive relationship to stock market movement". In: *Quantitative Finance* 15.10 (2015), pp. 1637–1656.