Spring 2012

Cloud Information Summarization With Mobile Interface

Hrishikesh Paranjape
San Jose State University

Cloud Information Summarization With Mobile Interface

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment of the Requirements

for the Degree

Master of Computer Science

By

Paranjape, Hrishikesh

May 2012

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Cloud Information Summarization with Mobile Interface

by

Paranjape, Hrishikesh

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Tseng, Department of Computer Science 05/21/2012

Mr. Yashodhan Deshpande, Infineta Systems 05/21/2012

Dr. Soon Tee Teoh, Department of Computer Science 05/21/2012

Acknowledgement

I would like to thank Dr. Chris Tseng for his excellent guidance throughout this project work and my committee members, Mr. Yashodhan Deshpande and Dr. Soon Tee Teoh for their time and effort. Also, a special thanks to my family and friends for their support.

Abstract

There is a large driver for making existing data accessible everywhere, not just for people who happen to be at connected computers. This project aims to design, develop, and test Information Summarization System with its mobile application, which will allow end user the quick mobile access to underlying data on the web. The mobile application allows users to search for information about topic of their interest. This application will make use of existing established search engines to summarize the information on requested topic. The computation will be done on the cloud and communication between client and server will be done via RESTful web services. The server crawls for the data and uses classifier to classify it into several classes. Several other open source technologies are used in this project. As a basic use case, client will send a keyword to server, and server will return classified data to the client. This classified data will be useful in forming summary of underlying data that server crawled. This project is an attempt to bring information closer to the user. The user can now access the summary data on the fly by making use of mobile application provided.

Table of Contents

List of Figures

List of Tables

1. Introduction

In this era of cloud computing, there is a large demand for accessible information. People want meaningful information out of large amount of available information. This demand is not just limited to those who happen to be connected to computers but also to those who use mobile devices.

The numbers of users using the social networking websites are growing exponentially. More and more people have started using social networking as a medium to express their opinions. In recent news, the number of users using Facebook (a leading social networking website) is found to be over 900,000,000 which, is close to 1 billion. So, that means every one in eight heads in the world is using at least one social networking website. In a recent study [1], information from twitterers (users of twitter) was found to be useful in co-ordination of medical work in disaster response. There are several other studies which have proved that the information from these social networking websites can be very useful in variety of real world domains such as marketing, disaster management, review generation, exit polls etc. My project focuses on tweets from Twitter; it will be very useful for users to browse quickly through the classified information and draw their conclusions based on classes. This classified information will further lead us to summary of the information. Moreover, users will be able to access this information from their mobile devices.

Information classification is usually done by extracting the most important words from the available data. This is done by breaking available text into words and sorting those words based on their frequency. Next, we have to tag that data to a predefined class. These classes can be anything, such as positive, negative, old, new, objective, subjective. In our case, a tag will be associated with a tweet. But, before we can assign a class to a tweet, we must have some data which can tell a computer program how to classify the incoming data. So, as an example, we need a word that is already known to be “positive”, and a sentence

which is “positive” because that chosen word. This is called as training data. Every classifier requires some training data in order to classify real data. This process will ultimately render us the several tweets assigned with a class for each tweet.

In short, the functionality of the system would be a user will input a keyword, for example, say iPhone, and in return it will get result of how many people are talking positive, and how many people are talking negative about iPhone. To accomplish this task, I have to make use of several server side and client side web technologies.

Of course, the user will be using a web browser or a mobile device; hence my client side has to a web application and/or a mobile application. There are several ways to develop my mobile and web client. My mobile client can be a native application for some platform like iPhone, Android, Windows Phone, Blackberry etc. It can also be a mobile web interface developed using some JavaScript library, there are several options, and we will see them in next section of this report. I also have developed a web application which will act as a client for user. On the back-end, we need a web service to interact with the client. We can write very lightweight web service in PHP. Off course, there are several server side options to write a web service, which we will discuss in the next section of this report, but we choose PHP. This PHP server side module is responsible for extracting the tweets from Twitter.

There is another important module in the back-end, data classification engine, it is heart of the system. It will classify tweets extracted by server side PHP module, and return it the result. The result will contain the tagged tweets which will be displayed to the users.

So, what is so special about this system, there are several similar social networking data mining tools available on the internet [2]. So, later in the report, we will discuss what more my project provides in addition to some of the already known twitter classification engines. For example, currently available tools such as Splunk do not categorize slang tweets [3]. On twitter, we find a lot users using slang in their tweet text. Also, later, we will analyze the system on speed, correctness, and scalability.

In the “Technologies” section, I will discuss several technologies used to accomplish the task and their comparison. In “Design and Implementation” section, I will discuss more about how I used those technologies for this project. The next one is “Analysis”, in which, I will analyze my system. And lastly, I will conclude the report.

2. Foundations

This project expects prior knowledge of some theoretical concepts. Those concepts will be discussed in this section.

2.1 Naïve Bayes Classifier

At the heart of this project is Naïve Bayes Classifier. Classifiers are used in classification of unstructured data. Among the classifiers available the simplest one is Naïve Bayes classifier. This simplicity of Naïve Bayes makes it usable in large number of domains. The Naïve Bayes classifier works on a simple but intuitive concept [4]. Also, it may outperform some complex classification algorithms in some use cases [5].

It is based on Bayes rule of conditional probability. It uses all attributes in the data and analyzes them individually, independently, and equally. As an example, training data consists of animals (say cats, fish, and birds), and our classifier has to classify any new instance of an animal. We know that action birds have wings, they can fly, etc. Also, cats have four legs, they like milk. Then, fish swim, and have fins. The Bayesian classifier will consider each of these attributes separately when classifying new instance of an animal. So, Bayesian classifier will not check if it has fins and if it swims, classifier will separately consider if the new instance swims, or a new instance have fins. After considering every attribute it will determine the probability of the new instance being a fish, a cat, and a bird.

In Naïve Bayes classifier, the probability of a new object of being in a class is determined as follows:

1. First we define prior probability. Prior probability of a class is ratio of number of instances of class in the training data to the total number of instances in the training data. In above example,

   Prior probability of cats = number of cats / total number of animals.

2. Next we define Likelihood of an instance for a class. It is the ratio of number of class instances in vicinity of instance under examination to the total number of instances of class. For example,

   Let “I” be any instance of unknown class

   Likelihood of “I” for cats = number of cats in cluster if “I” / total number of cats.

3. Then we define Posterior probability of “I” (any instance of unknown class) for any given class. It is product of prior probability of “I” and likelihood of “I” for a given class. For example,

   Posterior probability “I” for cats = Prior probability of cats * Likelihood of “I” given cats.

4. Likewise, we compute posterior probability of new instance for every class. In the conclusion, we determine the instance to be of class for which its posterior probability is greatest [5].

Above four steps will give us the class for a test statement (tweet). Now, To put it into mathematics [6]:

We want to calculate:

prob (tag fits|this tweet, and training data)

Using Bayes theorem:

*prob(tag|tweet, training) = prob(tag|training) * (prob(tweet|tag,training) / prob(tweet|training))*

Here,

prob(tag|tweet, training) = probability that given tweet should belong to a tag

prob(tag|training) = prior probability

prob(tweet|tag,training) = probability that words in input tweet appear in tag’s data set

prob(tweet|training) = how much each word in input appear in all training data

prob(tweet|tag,training) / prob(tweet|training) = posterior probability

2.2 REST Architecture Style

REST stands for Representational State Transfer [6]. It is a style of Software Architecture for distributed systems. REST is new age predominant web service design model.

The key goals of REST include:

- Scalability of component interactions
- Generality of services
- Independent deployment of components
- Intermediary components

In this project we use RESTful web services. RESTful web service is a web service implemented using HTTP. It has four aspects:

1. The base URI
2. Media type such as JSON, XML
3. Set of operations such as GET, PUT
4. API must be hypertext driven

3. Technologies

To start with I would like to enlist three main modules in my system, and explain the choice of existing framework for each.

1. Front End Web and/or mobile
2. Back End Web Services
3. Back End Classifier Module
4. Data Visualizations

3.1 Front End Web Technologies

On the front end, we need to consider presentation and data visualization aspects of the system. As the project focuses on Mobile Interface we will start with choices for platforms for mobile interface.

1. **Native Application Development for iPhone, Android, Windows Phone etc.:**

   If we choose this approach, we need to develop different versions of the same application, which is a tedious job to do. The market share of these platforms is changing too frequently to choose any one or some of them, so, by not choosing any one or some of them we are making our system not accessible to everybody, which defeats the purpose of making data accessible to anyone anywhere.

2. **Titanium Appcelerator:**

   Titanium Appcelerator is a platform which allows developers to deploy one application on multiple platforms, like iPhone and Android. Using this approach we can deploy the same application on iPhone, Android and Windows Phone 7. But, large learning curve, relatively small user community are some drawbacks of this technology.

3. **Mobile Web Application:**

   Considering that the web counterpart of mobile application is very desirable to develop, this approach can be useful in development of web interface as well. There are several JavaScript libraries which help us write mobile friendly web applications. Moreover, the power of new HTML5 and CSS3 allow us to style our web application in such a way that the application looks like a mobile application. The most used JavaScript libraries in this domain are sencha touch (mobile equivalent of ext js), jquery mobile (mobile equivalent of jQuery), moo mobile, etc. But, the look and feel of sencha touch applications is more like native mobile applications; also the user community of sencha touch is larger than that of jquery mobile. Hence, we go for sencha touch.

For client side web application, we will be mostly dealing with HTML markup, CSS styling. For dynamic content, there was an opportunity to learn other libraries like Ext JS and YUI, but both the libraries have proven to be heavyweight as compared to jquery. Also, jquery suffices specific requirements of the application. Hence, we choose jquery for dynamic content on the client side.

3.2 Back End Web Services

Our server side tweet extraction module communicates with the front end client application. This communication is carried out with the help of RESTful web services. There are several lightweight web service development platforms. Few of them are as follows:

- Zend PHP
- NodeJs
- Action Web Service (For Ruby on Rails)
- Apache Axis 2 – Java

For our use, I chose to write web services in PHP.

This back end PHP program is responsible for taking a search keyword from user, extracting tweets from twitter API containing the search keyword, passing the tweets over to python classifier, and giving the result from classifier back to client.

3.3 Back End Classifier Module

For our classification purposes we use Naïve Bayesian Classifier. In Python, there is a package available for Natural Language Processing, it is called as Natural Language Toolkit (nltk). Python nltk is collection of open source python modules, linguistic data, and documentation for research and development in natural language processing and text analysis.

There are several other implementations of Bayesian Classifier, but we use Bayesian classifier from nltk [7].

3.4 Data Visualizations

After classified data is sent to the client, we need to visualize it in form of interactive charts and graphs. For charting on the web browser, there are several JavaScript libraries, which can be used to draw charts on the client. We can also make use of few other API's for charting such as Google Charting API. We have following options:

1. Google Charting API:

   Google has a charting API that provides you the images of charts for a given input. Currently Google Charting API does only support static charting. So, it has a drawback of not having dynamic charts.

2. SVG based solutions:

   There are several SVG based JavaScript charting libraries available. Few of them are highcharts, raphael, and d3 [8]. An advantage of having an SVG based library is we can change the graph components on the fly without having to re-render it.

Another advantage is SVG based libraries are supported by all the main stream web browsers wiz. Internet Explorer (MSHTML), Mozilla Firefox (Gecko), Chrome and Safari (Webkit), hence we are less likely to face cross browser issues.

3. HTML5 canvas based solutions:

   Few other JavaScript libraries such as flot, sencha charts, and EJS are based on HTML5 canvas element. But, because of war of standards between mainstream web browsers, HTML5 canvas element is not yet supported by all the browsers. Hence we have to use excanvas.js in order to make canvas based libraries work with non-compatible browsers. What excanvas does is convert HTML5 canvas element to its VML equivalent.

Because of drawbacks of HTML5 approach and Google Charting API, we choose SVG based highcharts as our JavaScript charting library.

4. Design and Implementation

As described in earlier section, my project consists of 4 main modules:

1. Front End Web and/or mobile
2. Back End Web Services
3. Back End Classifier Module
4. Data Visualizations

In this section, above four main modules of the project will be explained in the order they were implemented.

4.1 Overview

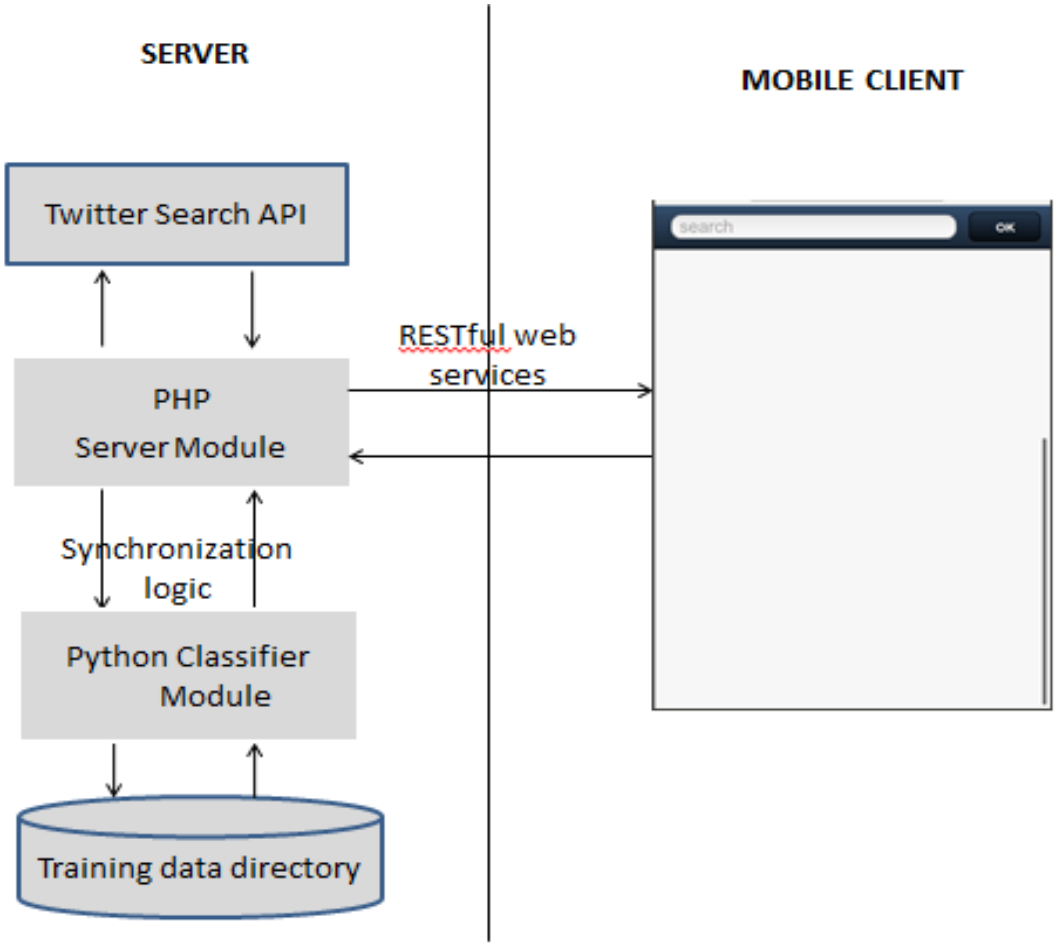

Figure 1: Application Overview

Above diagram shows the overall architecture of the system. As you can see the use case starts with user entering a keyword, PHP server module gets the keyword and sends the keyword to Twitter API, the Twitter API sends a list of tweets back as a reply of GET request. These tweets are then parsed and submitted to Python classifier module. Some synchronization logic is used while sending the tweets. This logic is explained later. The training data is a directory in which several files with are present, each file contains the training data for a specific class. After classifier module completes processing classification, it sends the results back to PHP module. PHP module does the job of parsing and converting before sending it to the client.

4.2 Classifier Module

At heart of the system is our classifier module. We use simple Naïve Bayesian classifier from Python's nltk library. Bayesian classifier needs training data. The training data is nothing but a set of statements with known tags. Based on those tags, classifier computes class for every incoming statement for which we have to find out the class. The process of classification was carried out in following steps:

1. Frequency analysis of training data:

   In this step, we parse the training data into words and find out frequency of each word. This frequency analysis is useful in determining probability of correctness of classification.

2. Extract word features from the data:

   After frequency analysis, we find out how much a word contributes in deciding the class of the subject. To create classifier, we need to determine which features are relevant. Feature extraction returns a dictionary that indicates what words are contained in the input passed.

3. Naïve Bayes Classifier:

   Python nltk has a classifier object that takes the training set and word features and gives back the class for the input statement. Correctness of the result depends on quality and quantity of training data chosen. Hence, it is highly important to choose precise training data. Following is the Python nltk NaiveBayesClassifier API.

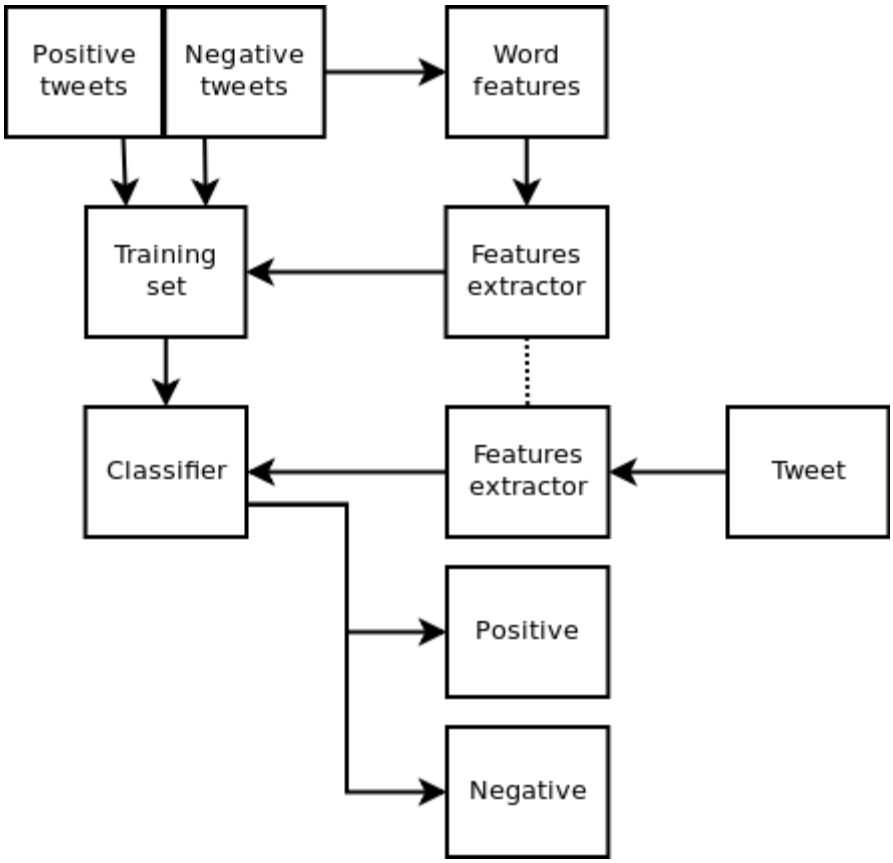

Figure 2: Python nltk NaiveBayesClassifier Block Diagram

[Image taken from laurentluce.com]

4. Training data directory:

   All the training data is kept inside a directory placed in the same directory where our python classifier program resides. The name of each file in the directory of training data is name of the class for statements contained in the file. For example, filename “positive” will contain all positive statements. My classifier program reads all files in the training data directory. So, every time when you want to add an extra class in

training data, you do not need to make any changes with the program, all you need to do is to add a file into the training data directory. Following is the screenshot of nautilus explorer showing training data directory.

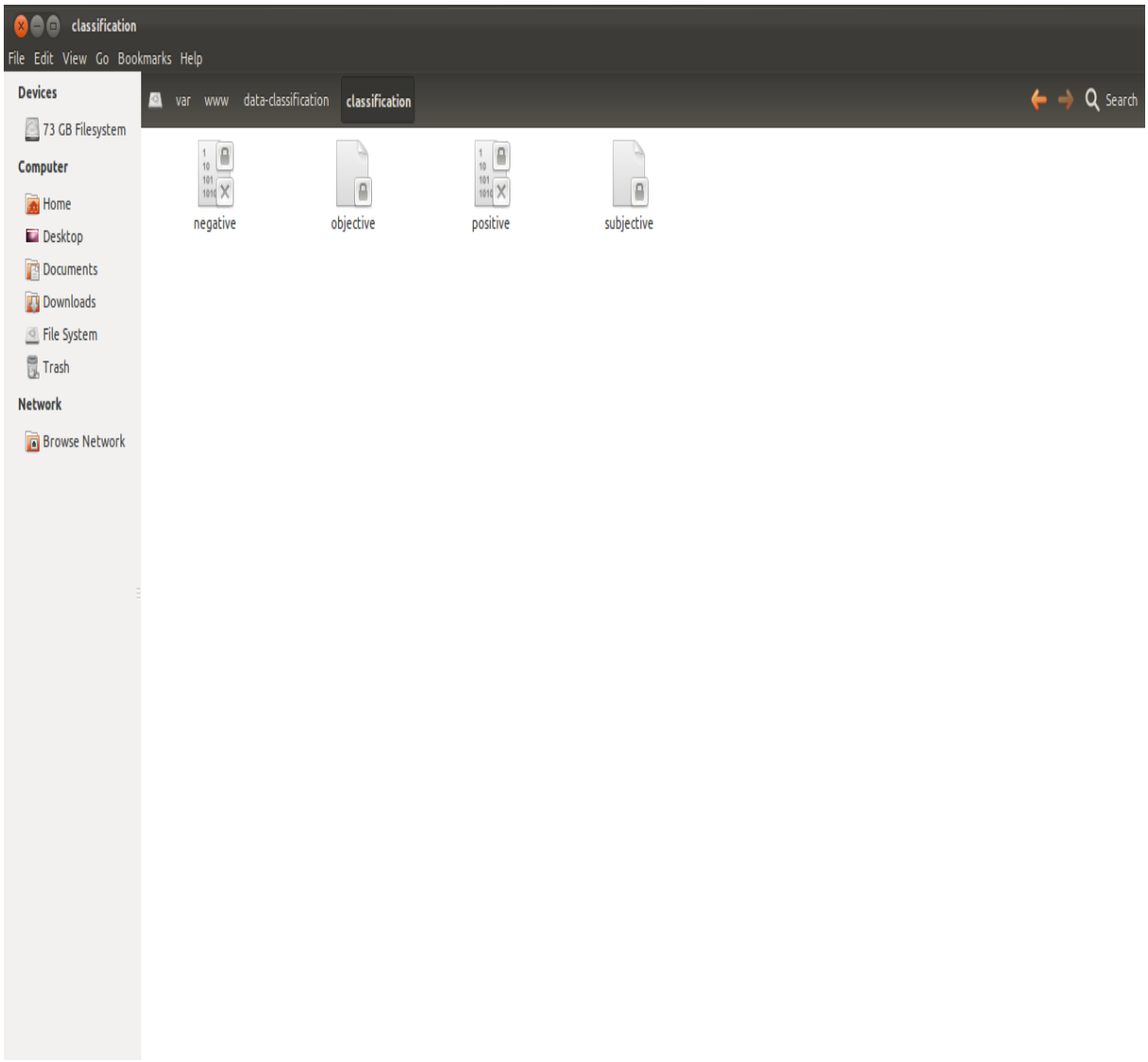

Figure 3: Screenshot of Training Data Directory Containing Training Files

Name of the file should be classname, and the text inside file should be a list of statements of that class separated by endline(\n) character.

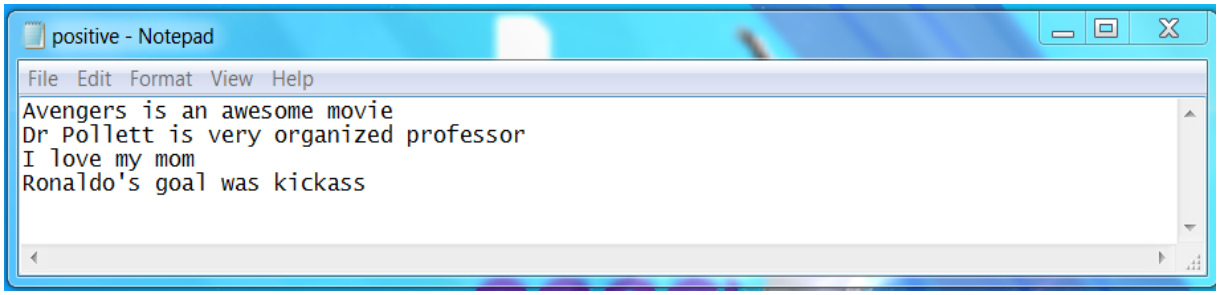

Figure 4: An Example of Training Data File

(File name: positive, Data format: end line separated)

I used few simple heuristics to increase accuracy and speed of classifier [9]. In the classifier module, when extracting word features, I can neglect the words with length 2 or less, because these words are less likely to contribute to the class of input (tweet/statement). Also, there are very few adjectives with word length 2 or less, hence it is safe to ignore the words having length 2 or less. I am also including slang language statements in the training data.

There were several performance issues with the classifier module; initially I was testing the classifier on my single core machine having 2 GB of memory. The training process of Naïve bayes classifier is very slow and took about half an hour (the exact analysis will be given later in this document) for 10000 statements of training data (I measure speed of a classifier in terms of number of lines it can analyze per unit time).

Fortunately, we have a solution; we can dump our trained classifier. Pickle library of Python is widely used to dump and retrieve the python objects. This way, we need to train our classifier only when there is a change in training data.

To save:

```
>>> import pickle
>>> f = open('my_classifier.pickle', 'wb')
>>> pickle.dump(classifier, f)
>>> f.close()
```

To load later:

```
>>> import pickle
>>> f= open('my_classifier.pickle')
>>> classifier = pickle.load(f)
>>> f.close()
```

Figure 5: Python pickle Library Sample Usage

Along with our training data, we also need to dump word features we extracted while analyzing the training data. We will be making use of those word features in order to find out class of a tweet.

The classifier program is listening to a file “inputdata”, “inputdata” is a text file written by PHP module. PHP module writes the tweets to this file. Once the classifier sees that the file in not empty, it parses the tweets from the file. Each tweet is fed to classifier to get a class attached to it. In the end we get a list of tuples with first field as tweet text and second field as class of the tweet. This tuple is then written to “results” file in csv format. Then it is responsibility of PHP module to send classified data to the client.

4.3 PHP Server Module

Like classifier module, PHP can also be divided into multiple parts as follows:

1. Web Service:

   Web service is used for communication between client side web application and the server; this web service takes a keyword from the client, and returns the classified tweets. In the meantime, the server has responsibility of extracting tweets from the Twitter API, pass them to classifier for classification, and get the results from the classifier. We use JSON data for convenience and speed. As JSON responses can

need no client side parsing and are smaller in size, JSON is better choice than XML for our application [10].

```
$.ajax({
    url:"localhost/classifier.php?q=iphone",
    method: "GET",
    success:function(response){
        //response is list of classified tweets related to iPhone
    }
});
```

Figure 6: jquery Web Service Caller (for web application)

```
Ext.ajax.Request({
    url:"localhost/classifier.php?q=iphone",
    method: "GET",
    success:function(response){
        //response is list of classified tweets related to iPhone
    }
});
```

Figure 7: sencha Web Service Caller (for mobile application)

2. Twitter API caller:

   PHP server is responsible for fetching the tweets from Twitter API. Twitter API takes a keyword and returns a list of tweets containing that keyword. Additionally, it can also take other parameters such as date range, number of tweets etc. which are quite useful in filtering the tweet data. Following is a sample twitter search API request string.

```
GET    http://search.twitter.com/search.json?q=blue%20angels&rpp=5&include_entities=
       true&result_type=mixed
```

Figure 8: Sample Twitter Search API Request String

You can choose the format of result, in our case we use JSON. As a new feature of twitter search API, it provides an option to retrieve popular tweets in addition to real time search results [11]. Following is snapshot of twitter search API reference.

Resource URL

http://search.twitter.com/search.format

Parameters

q required	Search query. Should be URL encoded. Queries will be limited by complexity. **Example Values**: `@noradio`
callback optional	**Only available for JSON format**. If supplied, the response will use the JSONP format with a callback of the given name.
geocode optional	Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by "`latitude,longitude,radius`", where radius units must be specified as either "`mi`" (miles) or "`km`" (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this `geocode` parameter to search near geocodes directly. **Example Values**: `37.781157,-122.398720,1mi`
lang optional	Restricts tweets to the given language, given by an ISO 639-1 code.
locale optional	Specify the language of the query you are sending (only `ja` is currently effective). This is intended for language-specific clients and the default should work in the majority of cases. **Example Values**: `ja`
page optional	The page number (starting at 1) to return, up to a max of roughly 1500 results (based on `rpp * page`). **Example Values**: `10`
result_type optional	Optional. Specifies what type of search results you would prefer to receive. The current default is "mixed." Valid values include: • `mixed`: Include both popular and real time results in the response. • `recent`: return only the most recent results in the response • `popular`: return only the most popular results in the response. **Example Values**: `mixed`, `recent`, `popular`
rpp optional	The number of tweets to return per page, up to a max of 100. **Example Values**: `100`

Figure 9: Sample Twitter Search API Reference

3. Communication with Python classifier:

   PHP server program also has responsibility of communicating with Python classifier. It first writes the tweets from Twitter API to a file named “inputdata” in the format classifier needs, then in waits for results from classifier module. Once it gets results from the classifier, it converts the results to JSON format and sends them to the client.

4.4 Client side web application

Client side web application was developed using jquery with highcharts as choice of charting library. The web application calls the PHP web service to get the classified tweets related to the passed keyword. As described earlier, the client gets the data in JSON format. This data is put into a table so that it is easier for user to see the results. To enchance the user interface, charts are also drawn in the application.

4.5 Client side mobile application

The mobile application was developed using sencha touch library. As discussed in the technologies section, the sencha touch application is a web application that just looks like a mobile application [12]. That is it is styled in such a way that it will look like a mobile application. The deployment process of such mobile web application depends on another open source tool called PhoneGap. PhoneGap is a tool which generates native applications for multiple platforms. These applications are empty applications with a “WebView” on it. “WebView” is a component that is available on every mobile platform. Later, we can just put our application on that “WebView” and deploy the application. Following are some screenshots of the mobile application.

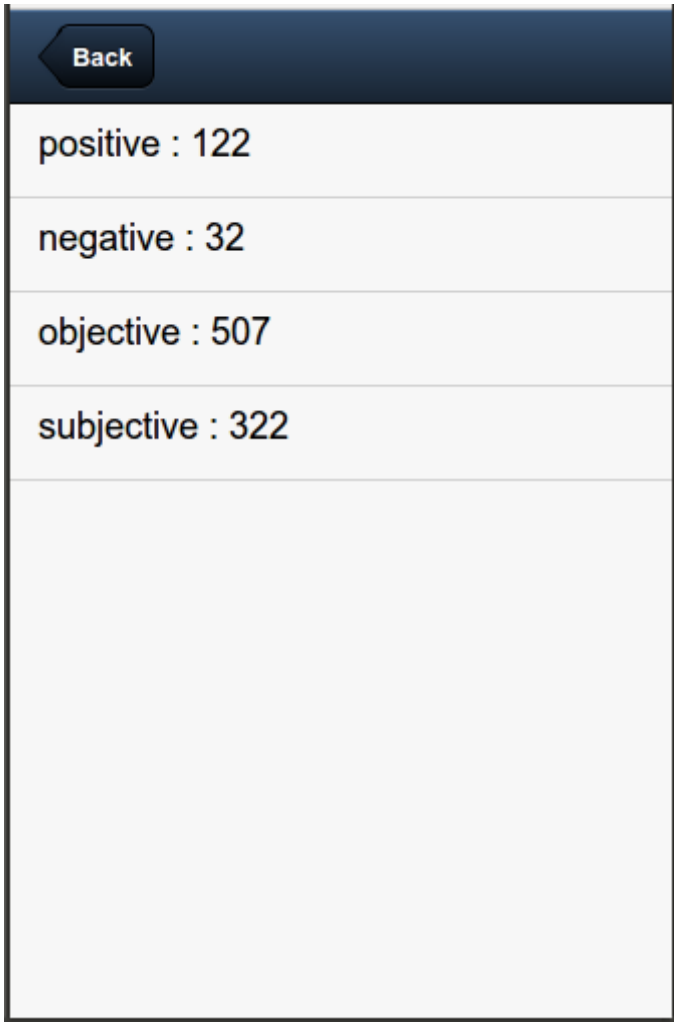

Figure 10: Mobile Interface Screen Showing Classified Tweets

Above screenshot shows how tweets are classified into classes. For storing results in UI we make use of sencha touch store API. Store acts as model in MVC design pattern in sencha touch. When Store is updated the view which is associated with the store, in our case the listview component is also updated. We also use HTML5 localstorage proxy for persistence of results. The results are stored in browser local history for future use.

Following screenshot shows how actual tweet data is displayed along with its class in the mobile application.

Figure 11: Mobile Interface Screen Showing Tweet Data With Class

4.6 Synchronization protocol

The Python classifier module is running in infinite loop and listening for changes in file “inputdata”. The disadvantage of that is the program has no way to know that the PHP module has finished writing tweets to “inputdata” file or not. Hence we have to design a protocol that will be obeyed by both the sides. This way our python module will know when exactly PHP module has finished writing the “inputdata” file. Following code snippets show how protocol is implemented.

```
        }
        $text .= "THISISLASTTWEET";
        file_put_contents($tweetFile, $text, FILE_APPEND | LOCK_EX);
}
```

Figure 12: PHP Module Side Implementation of Synchronization Protocol

The implementation of protocol consists of two steps. On the PHP server side, while writing the “inputdata” file, at the end of file we write “THISISLASTTWEET”. Simultaneously, the python classifier module will check if this line is read or not. Once python classifier module reads this line, it can assume that PHP module has finished writing tweets to the “inputdata” file.

```
while(True):
    all_tweets = open("inputdata").read().splitlines()
    if len(all_tweets)>0:
        if all_tweets[len(all_tweets)-1]=="THISISLASTTWEET":
            break
```

Figure 13: Python Classifier Module Side Implementation of Synchronization Protocol

On Python classifier module side, control exits the infinite loop only if it encounters last line of “inputdata” file to be “THISISLASTTWEET”.

In order to debug issues in synchronization of PHP web service module and python classifier module, we use debug messages on both sides. For the PHP module we store debug messages in a file, and for python classifier module we print messages in running console. Each log message has been given an identification number, which helps in tracking down the bug. Following are some screenshots showing debug logs for PHP module and python classifier module.

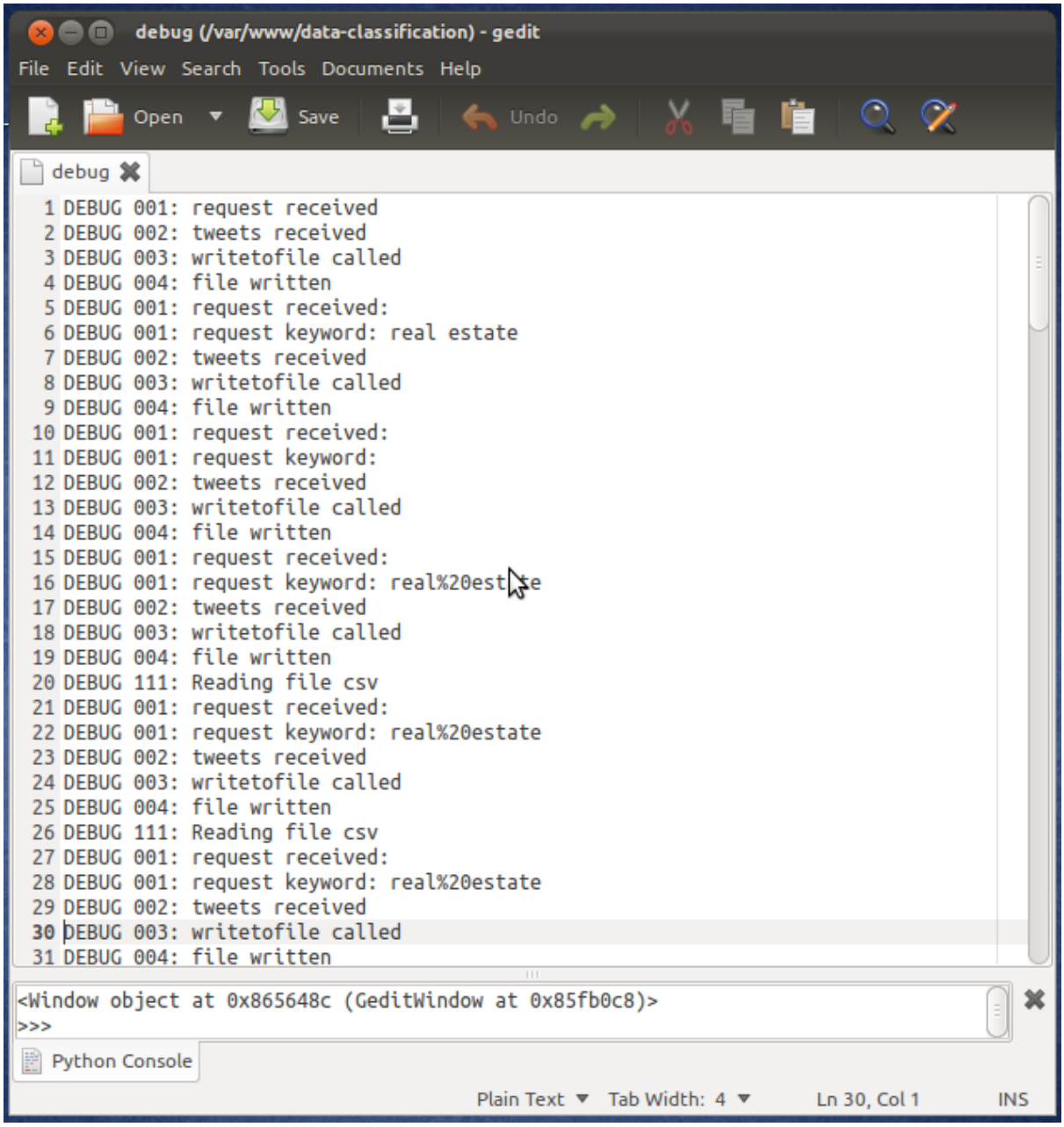

Figure 14: Debug Log File for PHP Module

Based on which logs are printed in the debug file, we can know the exact location of bug/malfunction in the program. The PHP module executes only when server receives a get request. After it receives request, it extracts tweets and writes them in “inputdata” file. Then, it goes into infinite loop until it receives any reply from the classifier module. Lastly, it converts csv reply from classifier module into JSON data, and sends it to the client.

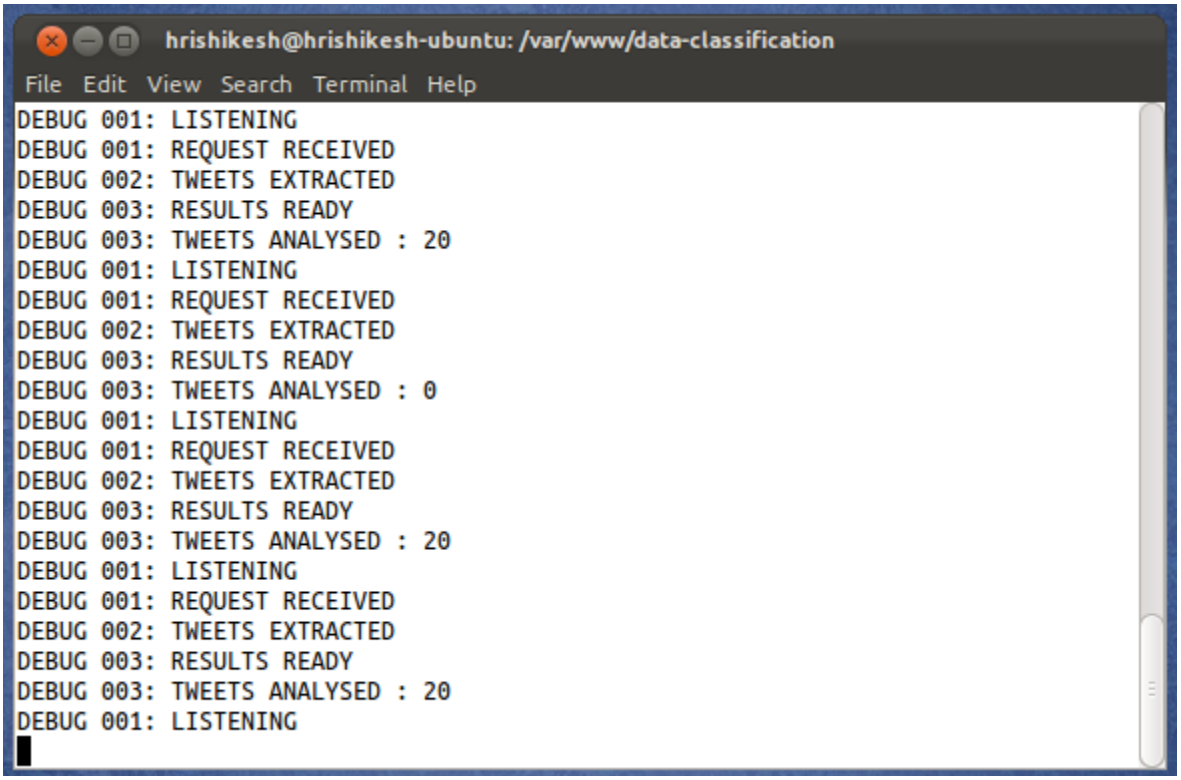

Figure 15: Debug Logs for Python Classifier Module

Python classifier module runs in infinite loop, listening for changes in “inputdata” file. We keep this python program always running. When “inputdata” file is written, python program starts processing tweets from that file. When it writes results into “results” file, it cleans up “inputdata” file and waits for PHP program to write next block of tweets into it.

4.7 Results and analysis

4.7.1 Performance Analysis

After building the system, I performed some experiments on it. My first experiment was to calculate time taken by trained classifier in analyzing tweets. For that purpose, I used time library in python. By changing rpp value in the twitter API, I changed the number of tweets my classifier module gets. Then I computed time required in seconds for the classifier to analyze those many number of tweets. I got the following results.

Number of Tweets	Time Required in Seconds
1	**0.91**
2	**1.76**
3	**2.33**
4	**4.28**
10	**10.44**
20	**20.05**
100	**94.76**

Table 1: Time Taken By Classifier to Analyze Tweets

The above table gives us the linear graph of number of tweets vs time taken. The experiment was performed on a single core 2 Ghz 32 bit Pentium processor. On a faster machine the results will be faster. The experiment shows that it takes about 1 second for the classifier to analyze 1 tweet. This is a long amount of time, and off course we do not want our user to wait for that much time. To nullify the effect of false positives and false negatives of the results of classifier, we need to analyze large amount of data. So, we must query for as much data as we can get from Twitter API. In order to do that we can use some tricks to query Twitter data. Also, on the classifier side, we can use some methods to make the classifier faster. The following graph shows the amount of time taken by classifier to analyze a number of tweets.

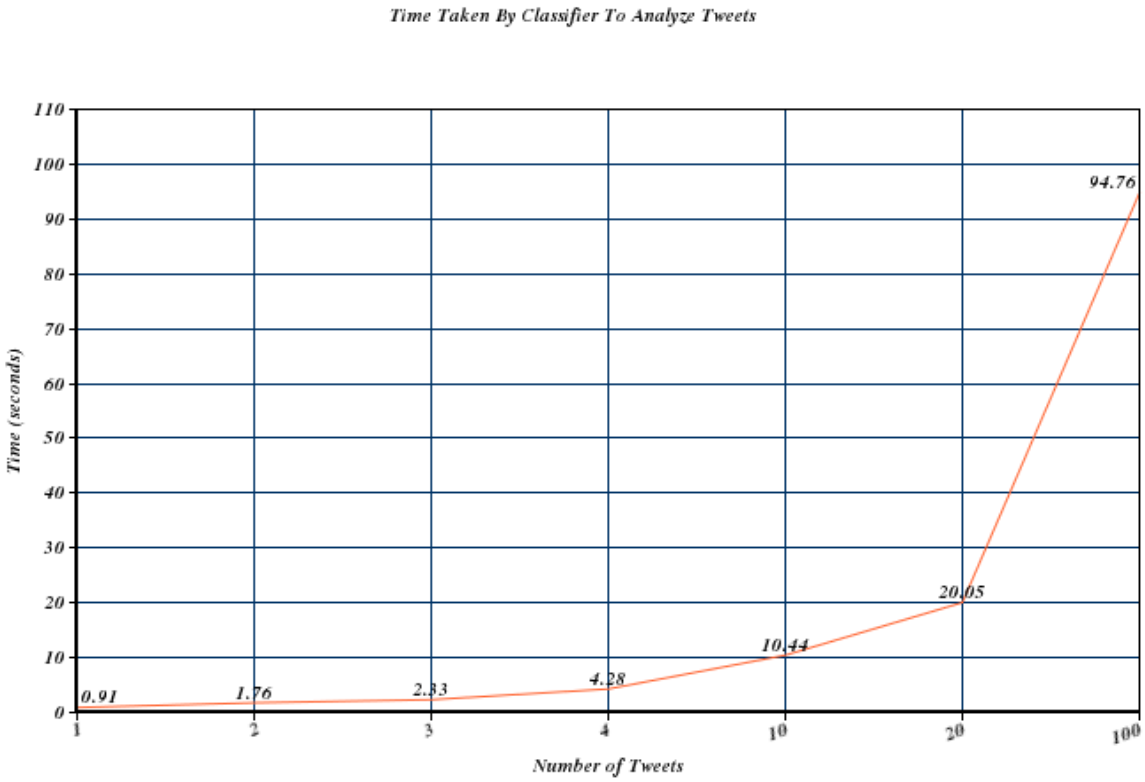

Figure 16: Graph of Time Taken Vs Number of Tweets

4.7.2 Fixes and Workarounds

Following are some ways to speed up the system:

1. The main reason behind this poor performance of the classifier is quality and quantity of training data. Hence, if we could manage to remove redundancies in training data, the performance of classifier will improve because data characteristics affect naïve bayes performance [13].
2. We can process tweets in small chunks. This way we do not keep our end user blocking. So, when user sends a keyword, after initial analyzed tweets, we can send new results automatically until user requests a new keyword. This will not speed up the classifier, but it will surely improve the user experience.

3. As specified earlier, the experiments were performed on a single core machine. Moving the classifier to faster and/or parallel processing will definitely improve the performance.

4.8 Application Flow Revisited

The application flow starts with the client application and ends with results shown in mobile interface on the client application. Complete flow of the application is summarized in following steps:

1. Client requests for the keyword
2. PHP server receives the keyword and extracts the tweets from twitter API
3. Extracted tweets are parsed and written in a file
4. These tweets are read by Python classifier module
5. Classifier classifies the tweets and tags a class to a tweet
6. Classifier module writes the result back to a file
7. PHP server reads the results and parses into JSON format
8. The JSON result is given back to client
9. Client application performs the job of displaying and charting the results

4.9 Client Detection

In modern UI techniques, client detection is practiced everywhere. If user opens the application on a mobile device, we show mobile version of the application, and if user opens the application using desktop browser, we show browser version of the application. Also, tablet version of the application can also be supported.

To achieve this, we use some available functions in Sencha Touch. In Sencha Touch, Ext.is.Phone is a global Boolean object which sets to true if the client is a mobile browser, otherwise it is false. By making use of this object, we can navigate user to desktop version of the application if Ext.is.Phone is false, otherwise we show mobile version.

5. Conclusion

In this paper, we have explored some simple data classification concepts, attempted to classify and summarize the cloud data from the Twitter API. We used some latest technologies such as RESTful architecture style, mobile web application using HTML5 and css3 etc. We also provided functionality of analysis of slang language. We also used several design patterns during the implementation of different components in the system. Lastly, we analyzed the system's various components.

The results show that my application gave all types of tweets related to keyword including real-time tweets along with their classification which leaded to summary of the cloud information. The application is made scalable by use of training data directory; hence if more classes are to be added to the system, there will be no change in the design. This application can prove to be grounds for further research in data classification and summarization.

References

[1] Aleksandra, Palen, White, Starbird, Bagdouri & Anderson, (2010). “Beacons of Hope” in Decentralized Coordination: Learning from On-the-Ground Medical Twitterers During the 2010 Haiti Earthquake: http://www.cs.colorado.edu/~palen/Home/Crisis_Informatics_files/Sarcevic-et-al-HaitiMedicalTwitterers.pdf

[2] Barone, Lisa, “5 Free Ways To Track Twitter Sentiments.” Smallbiztrends.com: http://smallbiztrends.com/2010/03/tracking-twitter-sentiment.html, Mar. 23, 2010

[3] “The Splunk Guide To Operational Intelligence”: Splunk http://www.splunk.com/web_assets/pdfs/secure/Splunk_Guide_to_Operational_Intelligence.pdf, [2012]

[4] “Classification Methods: Supervised Methods”: http://www.d.umn.edu/~padhy005/Chapter5.html, May. 17, 2006

[5] J. Huang, J. J. Lu, and C. X. Ling. *Comparing Naive Bayes, Decision Trees, and SVM with AUC and Accuracy.* ICDM 2003

[6] “Representational State Transfer.” Wikipedia: http://en.wikipedia.org/wiki/Representational_state_transfer [May. 4, 2012].

[7] Bird, S, Klein E., & Loper E. (2009). *Natural Language Processing with Python.* USA: O'Reilly Media

[8] Bostock, M. ***“****Visualizing Data with D3.js”* Presentation: SVG Open 2011

[9] Rennie, J. D., Shih, L., Teevan, J. & Karger D. R. *Tackling the Poor Assumptions of Naive Bayes Text Classiers.* In Proceedings of the Twentieth International Conference on Machine Learning.2003 , pp. 616-623.

[10] “JSON: The Fat-Free Alternative to XML”. Json.org: http://www.json.org/xml.html

[11] “GET search”. Twitter Developers Site: https://dev.twitter.com/docs/api/1/get/search [Apr. 18, 2012].

[12] “Sencha Touch 2: Build Mobile Apps with HTML5” Sencha: http://www.sencha.com/products/touch/features/ [2012].

[13] Rish, I., Hellerstein, J. & Thathachar, J.*An analysis of data characteristics that affect naive Bayes performance.* ICML 2001