TWITTER TRACKER & ANALYSIS

Luke PRITCHARD n9460250

Lachlan POND n9475095

Assignment 2 – CAB432

Dr Jim Hogan

1/1/2017

Introduction

The Twitter Tracker & Analysis is a web application design to help users track tweets streamed from Twitter. Users can input multiple hashtags or keyword of the tweets they would like to track. This will cause a livestream from Twitter, constantly receiving new tweets that matches the user's inputted keywords.

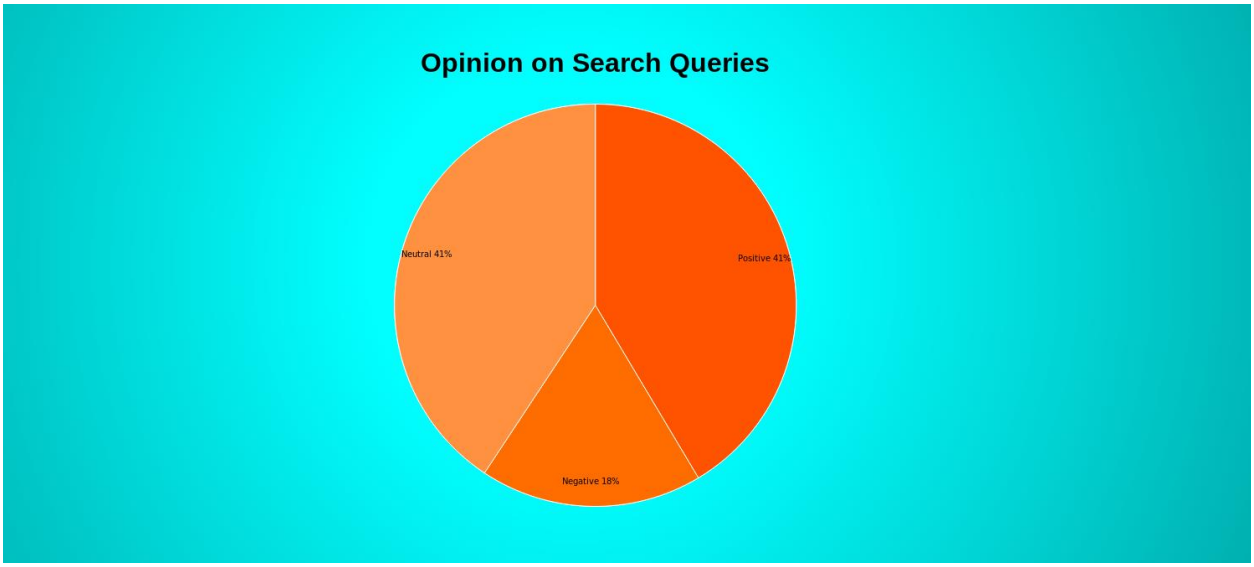
The output produces an aesthetic and easy to read list of tweets sorted from most recently tweeted to older tweets. The user then has the option to view all of these tweets (generally the most recent 15) or even go on to analyses these tweets via the pre-defined analysis page. Using the analysis page, the user can extract data from the tweets such as the ratio of positive to negative tweets, most recurring words across the tweets and the most recurrently mentioned country or city.

Use cases

Use Case 1

Steve is a reporter and he is reporting on a political event that is currently occurring. Steve would like to find out what the public opinion of Donald Trump is from twitter users during the event.

Screenshot:

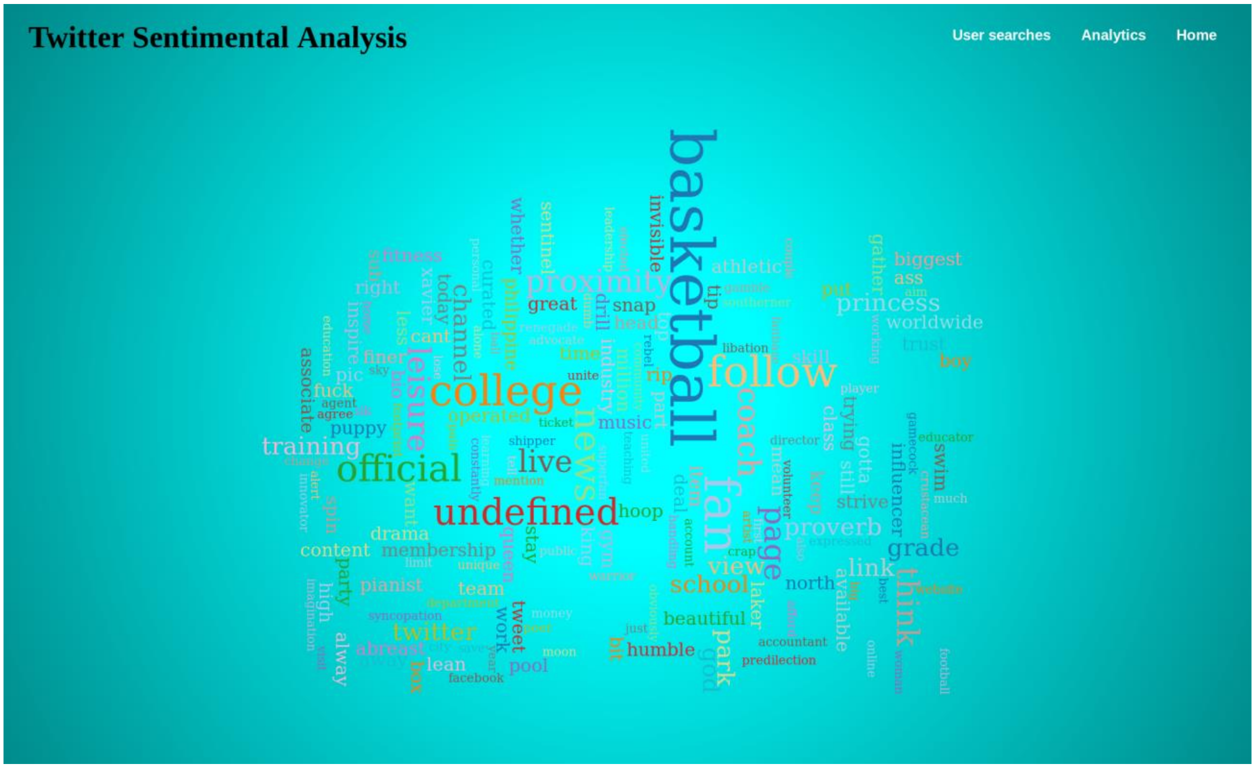


Steve is able to enter his query (Trump) on the homepage of the application and navigate to the analytics page where he will find an option to go to the public opinion of the entered queries. There, Steve can see the percentage of how many tweets are either positive, neutral, or negative. This was achieved using the Sentiment package for NPM and outputting that information to a .tsv file that is then read by the generatePieChart() function to generate a pie chart using the d3 visualization library.

Use Case 2

Brooke’s favorite sport is basketball. Brooke would like to find out about what people associate with basketball.

Screenshot:

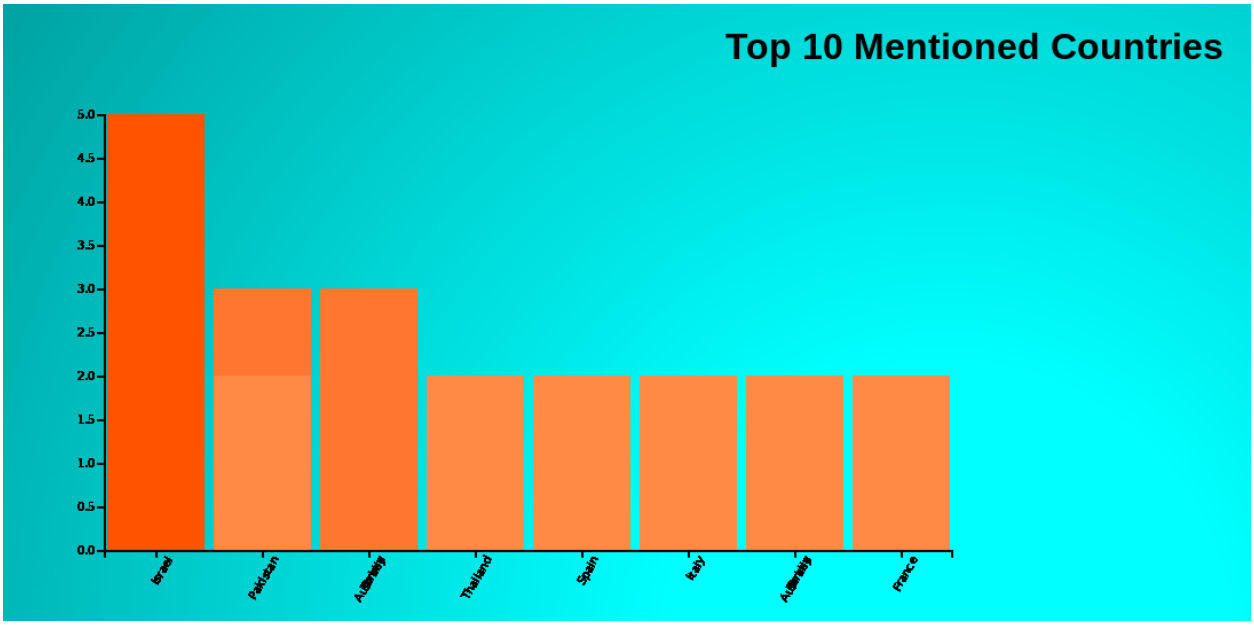


Brooke is able to enter her query (Basketball) on the homepage of the application and navigate to the analytics page where she will find an option to go to the word cloud of the current queries. There Brooke can see which words occur more often across the different tweets and therefore find what people commonly associate with her favorite sport. This is done using the d3 visual library.

Use Case 3

Matthew is a photographer and likes taking pictures of beautiful scenery and landscapes. Mathew would like to find out which countries might have the most beautiful scenery and landscapes for him to photograph.

Screenshot:



Matthew is able to enter his queries (beautiful, landscape, scenery) on the homepage of the app and navigate to the analytics page of the app where he will find an option to look at the most occuring countries. Matthew can see that Israel is mentioned the most in regard to the entered search query and might plan a visit soon. The information is found using the Cities-list and Country-data packages for npm. The data is then displayed using the d3 visual libarary.

Services

Application Programming Interfaces

Twitter API

The Twitter API is the applications fundamental base. On entering a string to search, the API is constantly streaming results and feeding them back into the user interface.

The Twitter API is declared using the following line of code:

```
var twitter = require('twitter');
```

We then establish the client to the Twitter using the provided keys as follows:

```
var client = new Twitter({
        consumer_key: 'arqa9nkL9XPIgHfnPGGc1qan6',
        consumer_secret: '57KUJIkZgdEUakKpLTQ7iLkHXXCef144IHZ4h0yVCcRpKMBs9R',
        access_token_key: '923715585267023872-UPiPS9fioX4v9urtFqFntMwck5acu1S',
        access_token_secret:  '0NRjoZBaNahTqGZsKrVNjFw41JJLneromZBn5LoqSNmTw'
});
```

With the client established the actual stream will need to be set up. This is the fundamental basis the application as this is continuously streaming data from Twitter. The Twitter stream parameters are set as follows:

```
var stream = client.stream('statuses/filter', {
        track: stringToSearch,
        language: 'en'
});
```

With the actual stream parameterized, to process the incoming data we use the following code below. Note there is a variable named message containing the required extracted information which will be added to an overall array stacked in JSON format. Within the message variable, the parameter dateSec is a variable with the value of the date in seconds. This is used for sorting purposes in the user interface.

```
stream.on('data', function(event) {
	var message = {
		name: event.user.name,
		username: event.user.screen_name,
		date: event.user.created_at,
		tweet: event.user.description,
		timestamp: dateSec
	}
	results.push(message);
});
```

Google Maps API

The Google Maps API is only on occasion used in the web application. As explained in the packages section of the report, every word in each tweet is parsed through a filtering system. This is used to check if the word is a valid English word, does not contain any numbers, also singularize the word so plurals and non plurals are not counted individually and so on. In addition to this, countries and cities are detected using these packages below also. If a country is detected, then it is added to a list of detected countries which is later used for analyzing.

Upon detection of a city, there is no known specific package that can return the country of a city. Thus as we require the country for our 'detected countries' page, the Google Maps API is used to return this.

The Google Maps API is declared using the following line of code:

```
var google = require('@google/maps');
```

We the establish the client by entering our API key:

```
var googleClient = google.createClient({
	key: 'AIzaSyCqJSEIN_kQHhmIO9-bBNA47Jhj-Wz-HLA',
});
```

The geocode application is the used to return the country and push it too the countries list as follows:

```
googleClient.geocode({
        address: word
        }, function(err, result) {
        if(!err) {
                var country = result.json.results[0].address_components[2].long_name);
                countriesList.push(country);
        } else {
                throw err;
        }
});
```

Packages

The web server prominently runs the Express JS web framework which sits atop of Node JS. Along with the Express JS framework, multiple JS packages from NPM are multiple node packages also used to maintain the service. These packages include:

- Express (web framework)
- Body-parser(NPM package for parsing middleware)
- @google/maps (NPM package for Google Maps API)
- Twitter (NPM package for Twitter API)
- Natural (NPM package for natural language tools)
    - Tokenizer (natural extension - splitting strings via case)
    - NounInflector (natural extension - singularized plural word converter)
- Tsv (NPM package for parsing and editing CSV and TSV files)
- Fs (NPM package for opening, reading and editing files)
- Sentiment (NPM package used for positive/negative detection)
- Country-data (NPM package for country detection)
- Cities-list (NPM package used for city detection)
- Check-word (NPM package to check if a valid English word)
- People-names (NPM package to check if a valid English name)

Technical Description

Client side

With the application predominately being written on the server-side, there is little to do on the client side. HTML5 and CSS3 represents the user interface on the client side, even though this is later manipulated on the server side.

JavaScript is used on the client side to display the data that was generated via server side code. For instance, when the index page is loaded the JavaScript code gets all the JSON data from the file location ‘./top15.txt’ and then parses this data to HTML output so the user can read it. The top15.txt file is a file which was created in the server side scripting. The following code shows how every time the page is loaded the data is grabbed:

```
onload = getData();
function getData() {
	loadDoc();
}
```

The function loadDoc() extracts the information from the top15.txt file and displays it in an aesthetic manner for the user to see. It grabs the data, extracts it and sends it to the div tag ‘messages’ using Ajax as opposed to refreshing the page every time.

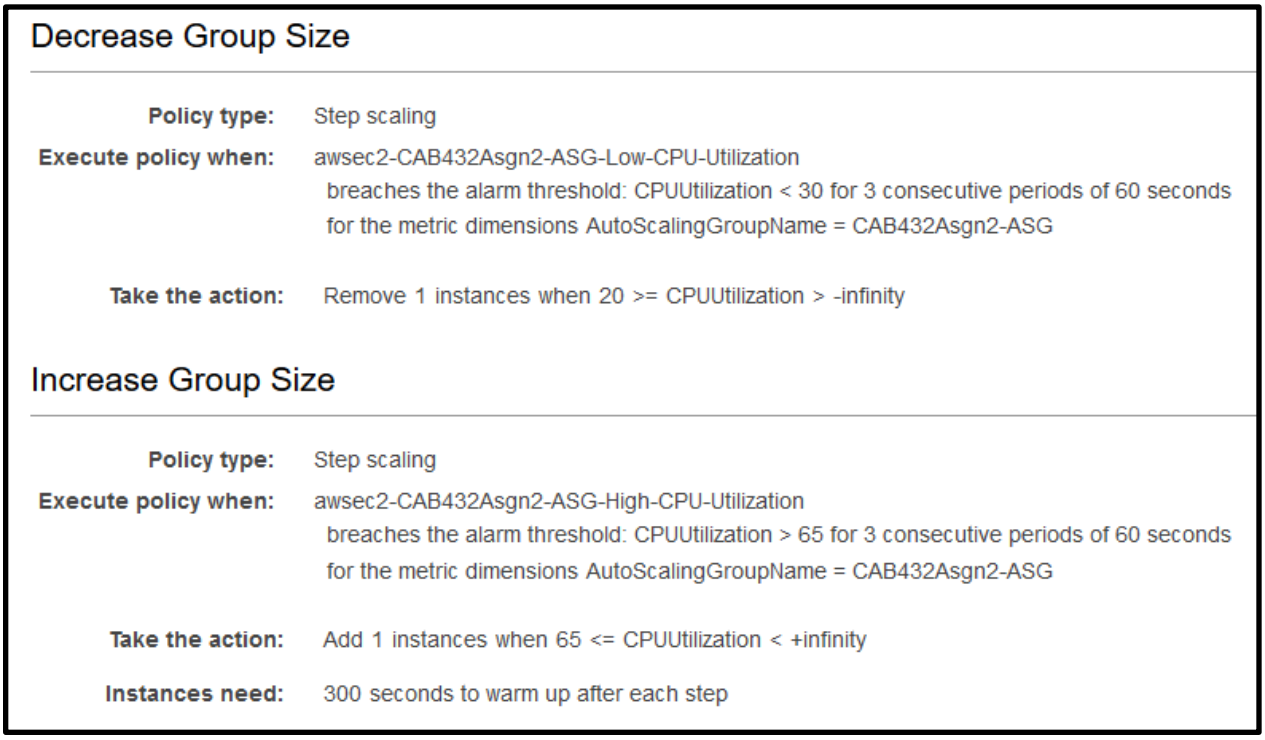
```
function loadDoc() {
	var xhttp = new XMLHttpRequest();
	xhttp.onreadystagechange = function () {
		if (this.readyState == 4 && this.status = 200) {
			document.getElementById(“messages”).innerHTML =
				this.responseText;
		}
	};
	xhttp.open(“GET”, “data/top15.txt”, true);
	xhttp.send();
}
```

We then use the following code to ensure the data is updated every second:

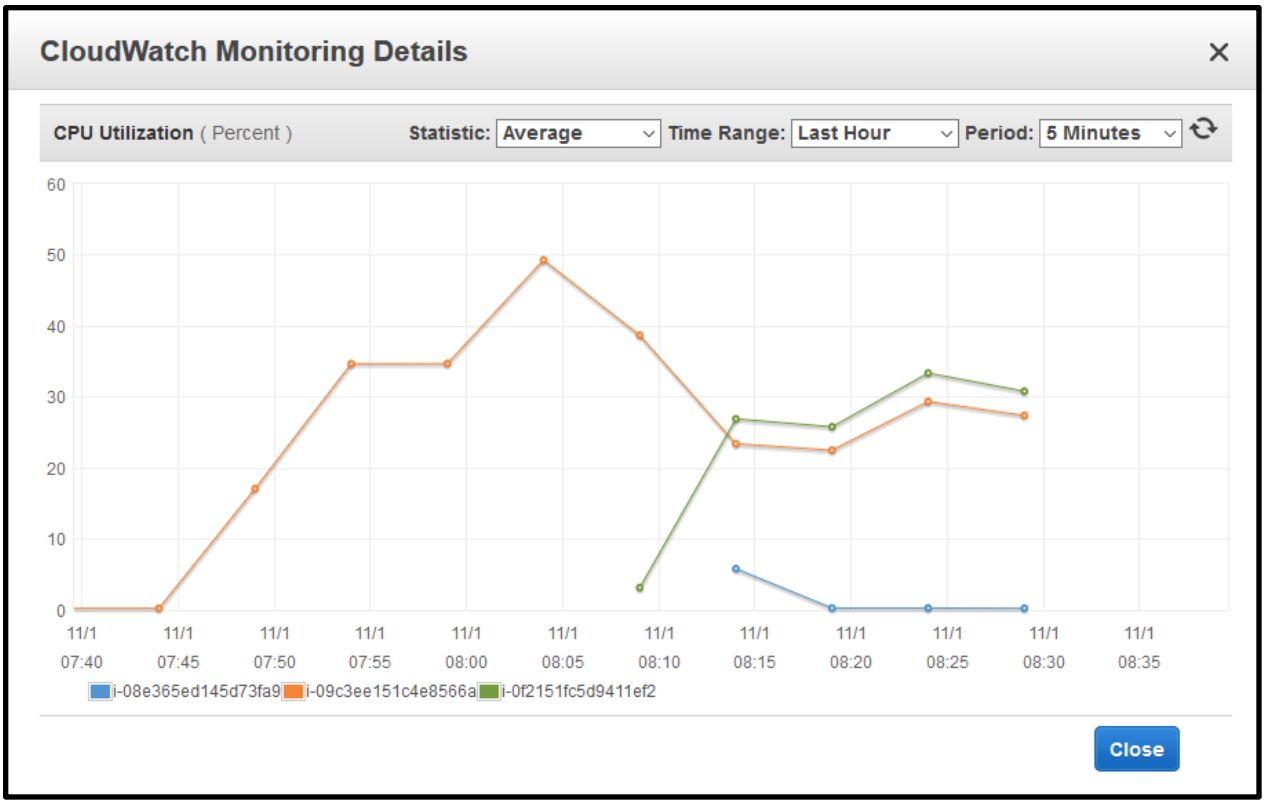
```
setInterval(function() {
	loadDoc();
}, 1000);
```

Server side

The server itself is in hosted on AWS and makes use of the npm package pm2 to start the server on instance startup without the need to ssh in and do so manually (pm2 is only a part of the AWS image and not in the provided code). An instance of the application running on an Ubuntu server was captured and saved as an Amazon Machine Image (AMI). This AMI was then used as a base instance to setup a launch configuration that could be used by the Auto Scaling group to help handle server load. The Auto Scaling group was setup with an initial one (1) instance with the minimum amount of instances set to one (1) and the maximum set to five (5).
The Scaling Policies were setup as shown below.



The setting were made quite generous since we have found it somewhat difficult to provide enough load at times. A classic Load Balancer was then created to delegate resources between the different instances as they were made and destroyed. The results of the combined efforts of the Auto Scaling group and Classic Load Balancer can be seem below.



Aside from the physical server itself, the tweets are parsed through packages in order to filter the responses and obtain the results from the stream. The first package that the result parses is the package named ‘Sentiment’.

Sentiment is a package that takes a string and returns double value based on its relevance to being a positive or negative string. If the value is a positive number then the string queried is positive, that goes for the same for negative. If the returned value is 0, then the string is considered neutral. We parse this through the sentiment package in order to obtain our ratio of positive-negative posts in our pie graph.

Natural is then used to begin performing queries on the individual words in the tweet. The package Natural and its module Tokenizor splits the resulted string into an array of words. This will allow us to cycle through each word in the tweet extracting specific information.

Each word in the string is checked to see if it a valid word, checking if it does not have any invalid characters or numbers that you not see in a word. To do this we use the standard library function parseFloat(), isNaN() and isFinite() as shown below:

```
if(isNaN(parseFloat(string[i])) && !ifFinite(string[i])) {
        // Continue
}
```

To ensure words are not duplicated unneccesarily for our word count, and that plural and non plural words are not counted indivisualy either, we use the standard library function toLowerCase() as well as the Natural packages 'nounInfector' module as shown below:

```
var newWord = nounInflector.singularize(string[i].toLowerCase());
```

Now that the word is established to be alphabetical and not numeric, we need to ensure this is in fact an English word and not garbage text. To do this we use the following NPM packages:

- Check-word (A dictionary that checks if inputted words are valid)
- People-names (A library of all known English names)
- Country-date (A library of all the countries)
- Cities-list (A library of all the cities in the world)

If the word is valid for one of these four packages then the word is a proper English word thus we can add it to our 'words list' in our analytics section. To check the word is valid we use:

```
word = string[i]
if(words.check(word) || names.isPersonName(word) || lookup.countries({name: word}) != 0
        || cities[word] != undefined) {
        listOfWords.push(word)
}
```

Lastly now that the word is a proper English word we run it through two more filters for country detection data. The first filter is very simple, it searches the country and if it is valid adds it to the list:

```
var country = lookup.countries({name: word});
if(country != undefined) {
        countries.push(word);
}
```

Secondly and lastly, if the word detected is a city, we parse the city through the Google Maps API to return its country and then add that to the list. The reference code for this can be found in the above section 'Google Maps API'.

Test Plan & Results

ID	Action	Expected	Appendix	Result
1	On load of initial index page	Blank page with search bar appears as single instance	Index page	Pass
2	On input of search string	Stream is connected to Twitter concurrently updating index page showing results	Index page	Pass
3	On input of search string	Concurrent results are show in in order of time posted	Index page	Pass
4	On input of multiple search strings	Stream is connected to Twitter concurrently updating index page showing results	Index page	Pass
5	On selection of analytics	Directed to ./analytics page showing all analytics of application	User searches	Pass
6	On selection of user searches	All the recent searches made by users shown	User searches	Pass
7	On selection of user searches	Results shown are for all instances and not singular instance	User searches	Pass
8	On selection of mode of words	Interactive file containing the top 30 words are displayed	Mode of words	Pass
9	On selection of positive/negative count	Pie graph showing the ratio between positive, negative and neutral tweets are shown	Positive/ negative count	Pass
10	On input of a new search string	Data recurred from previous search is voided and instance starts fresh	Index	Pass
11	Input of a jiberish string	No data from Twitter stream will be received	Index	Pass
12	No search selection positive/negative count	Circular pie graph with no data is shown	Positive/ negative	Pass

			count	
13	No search selection of mode of words	No data is shown in the file	Mode of words	Pass
14	On selection of mode of countries	Bar graph showing most popular mentioned countries	Mode of countries	Pass
15	No search selection of mode of countries	Empty bar graph is shown	Mode of countries	Pass
16	Refresh most occurring countries page	Bar graph containing the correct data	Mode of countries	Pass
17	Refresh public opinion of queries page	Pie chart should still display the correct data	Positive/ negative count	Pass
18	Refresh of word cloud page	Word cloud should still display the correct data	Mode of words	Pass
19	Resizing homepage	Data stays centrally aligned	Index	Pass
20	Resizing Analytics page	Data stays centrally aligned	Analytics	Pass
21	Resizing most occurring countries page	Data stays centrally aligned	Mode of countries	Pass
22	Resizing public opinion of queries page	Data stays centrally aligned	Positive/ Negative count	Pass
23	Resizing word cloud of Most Common Words page	Data stays centrally aligned	Mode of word	Pass
24	New Tick for input data	Bar Chart should update with the new information	Mode of countries	Pass
25	New Tick for input data	Pie Chart should update with the new information	Positive/ negative count	Pass
26	New Tick for input data	Word cloud should update with the new information	Mode of words	Pass

27	Terminating an instance	Create a new instance to replace the old if necessary	AWS	Pass
28	Heavy load	Start more instances to help handle the extra load	AWS	Pass
29	Connect through the load balancers DNS	Connect the user to an instance and operate as normal	AWS	Pass
30	Idle after load	Reduce the amount of instances if possible.	AWS	Pass

Possible extensions

As any application, improvements and extensions could always be inherited. If the allocated time to the web application was greater these extensions can definitely be implemented on a practical level. Some of the improvements and extensions are as follows:

Artificial Intelligence

If there was quite a substantial amount of time to create this application it would be possible to create some sort of AI program the automatically detects fluctuations within the tweets and navigate to the source.

For instance, if we search the words ‘disaster, Australia’ we would have an incoming stream of everything relating these words. However, with an AI implementation, the user could be notified if a fluctuation happened in the income of streamed results thus denoting that some sort of disaster in Australia is happening. Through training the AI will then be able to find out what the disaster is and where, and even notify the user.

Links to Twitter feeds

The application is very basic in the sense of it connects to a stream, creates automatic scaling to account for the load and performs analytics on the result stream. To better extend this, another page could be entered where if the user sees a tweet that he or she is interested in, the user can select that tweet and be directed to a page of information.

This page may be the exact tweet source on Twitter, or it can be another page that was made by the resulting JSON specify details about the tweet, the user who made the tweet and others who have shared or liked this tweet.

With this information we can then perform more analytics such as average number of shares per tweet, who shared from what location and so on.

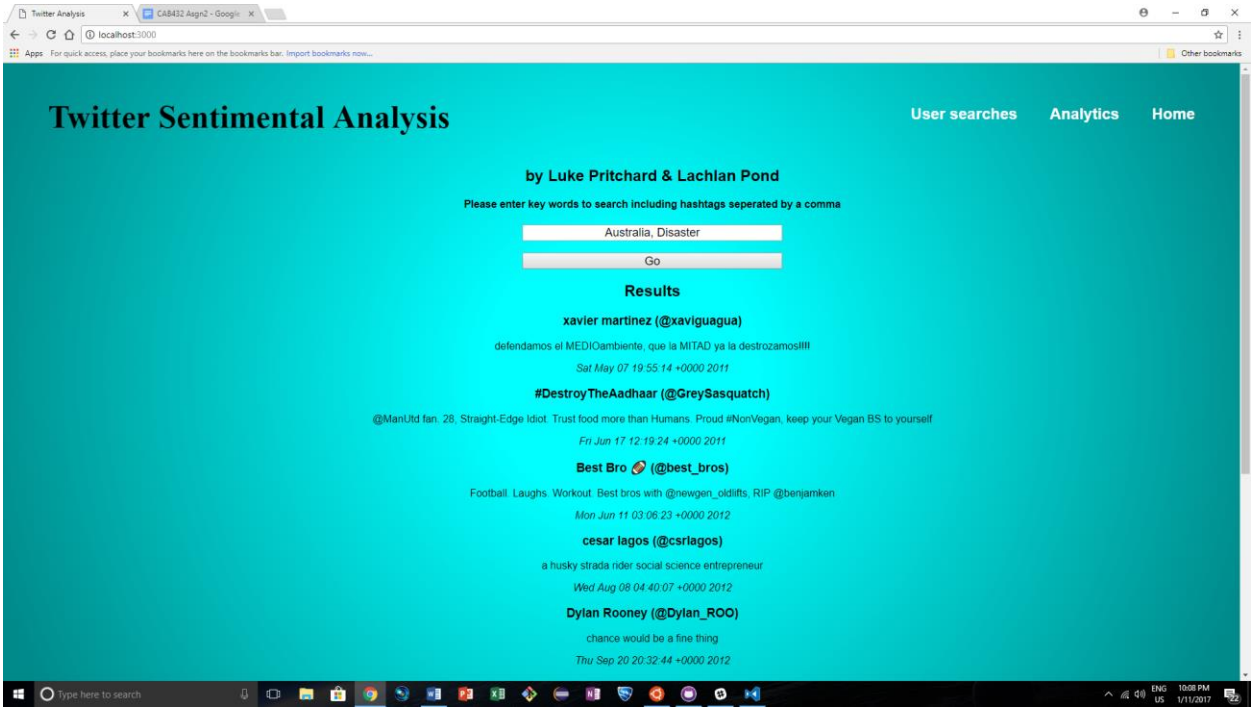
Advanced Search

Right now the application only allows the user to enter a string of single words separated by a comma to search. This will in all start a stream with Twitter feeds which relate to one or more of those words.

As a possible extension, we could further give the user a lot more options to add to their stream. This gives the application an ‘advanced search’ option. With this option, the user will then be allowed to stream from Tweets which matches exactly all the words entered, not just one. The user will also be capable of selecting the language of the streamed tweets. In addition, other parameters such as specific users, specific dates and even specific tweet lengths will be allowed to be selected if an advanced search option was implemented

Appendix

Index page



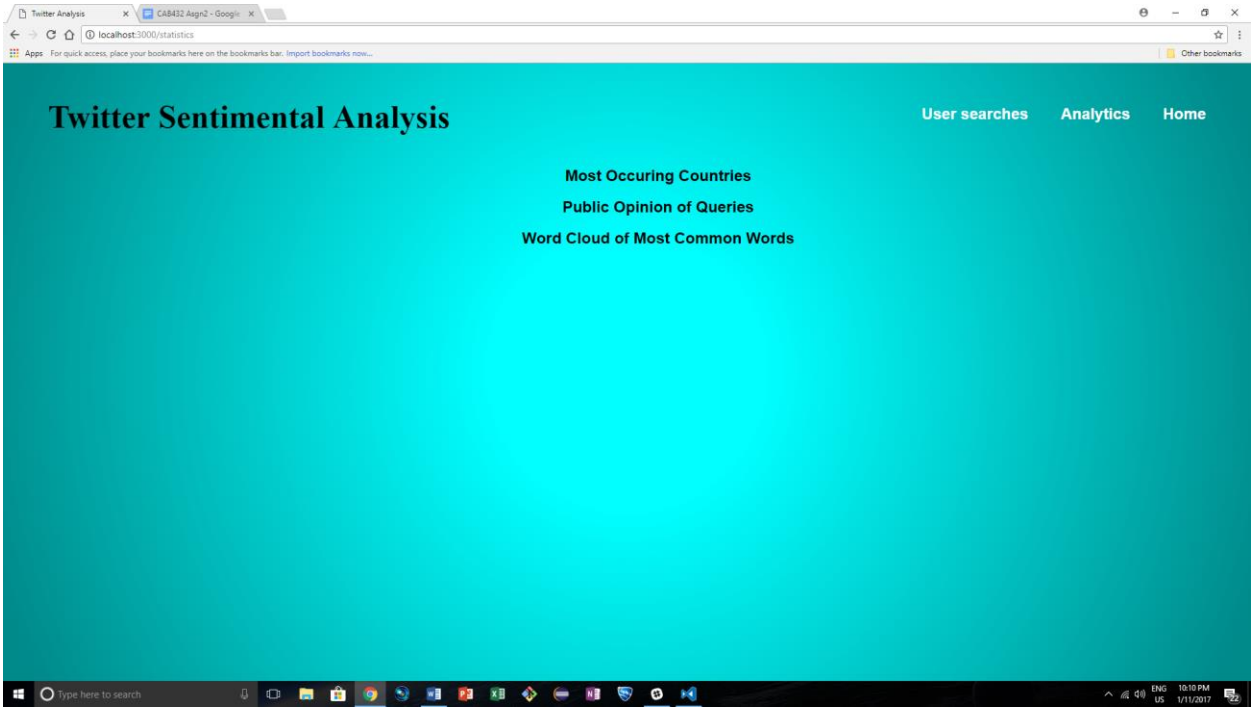
(1) Input field

The input text field is a field in which the user enters the string that he or she would like to match for the streaming. Upon entering their queries, the user can then start the stream of Twitter cheats by clicking the “Go” button.

(2) Tweet Results

The results list shows the returned tweets found by the stream with the given queries. The results are displayed with the username, time posted, Twitter handle, and the tweet content itself. The results list the most recent fifteen (15) tweets that the twitter stream has found and updates once per second.

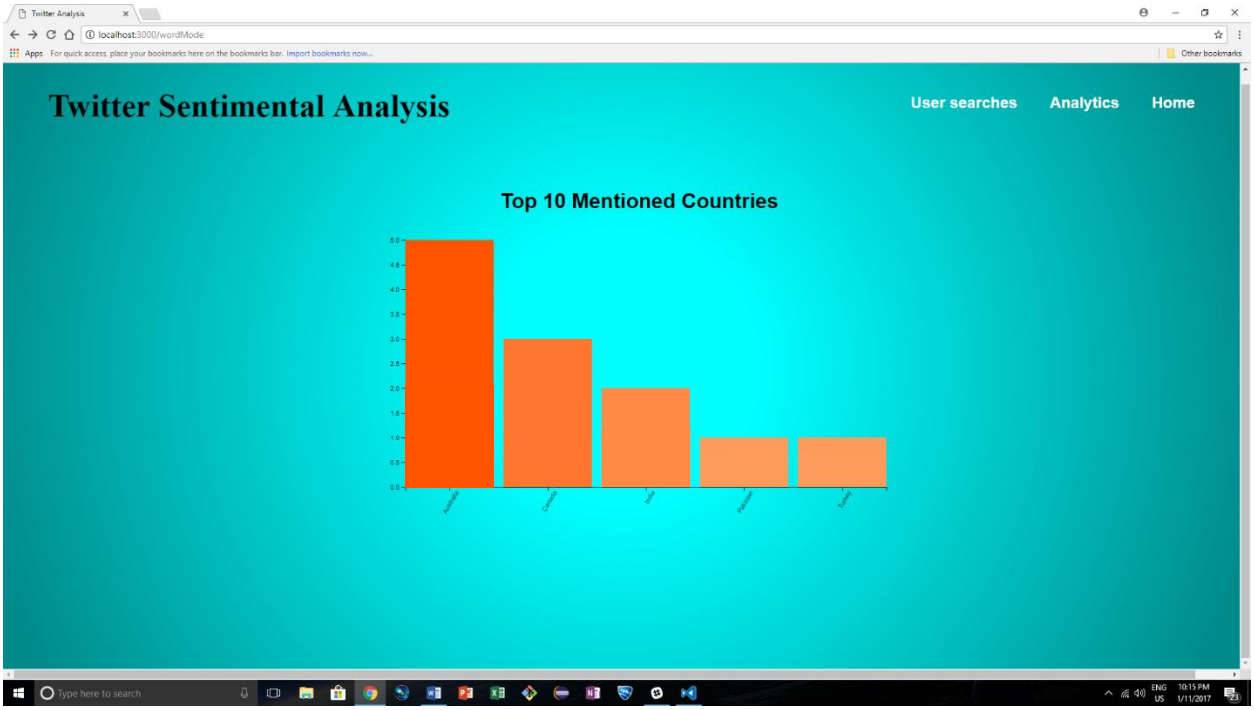
Analytics page



(1) Analytics List

1) Most Occurring Countries - The most occurring countries option will take the user to a bar chart of the countries which are commonly named in the Twitter stream
2) Public Opinion of Queries – A pie graph showing the ratio between positive, negative and neutral post
3) Word Cloud of Most Common Words – An aesthetic cloud gathering of words displaying the most common being a bigger word

Most Occurring Countries Page

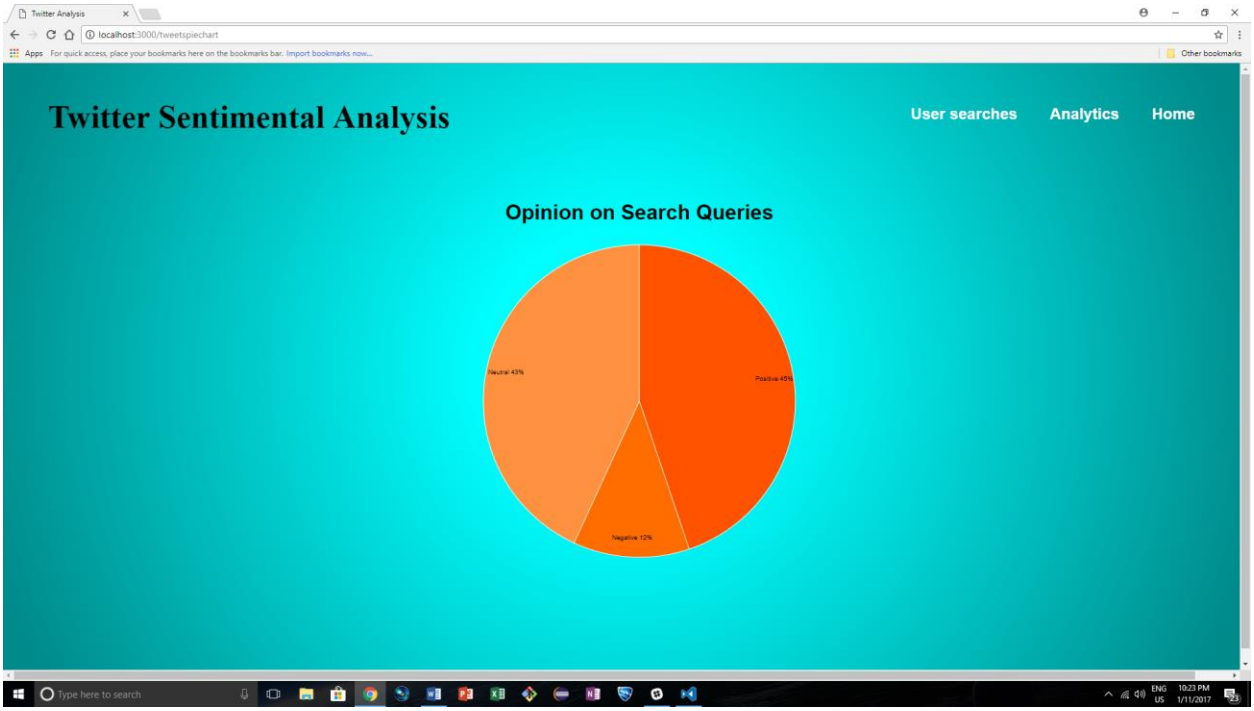


The top 10 mentioned countries is a list of auto detected countries and cities and the number of times they have been mentioned. In this example, as you can see from when inputting ‘Australia, Disaster’, of the tweets five of them mention Australia, three mention Canada and so on. Since only five countries are shown, the graph defaults to only showing the five as opposed to ten.

The package used to create this table is D3.js. D3 is a JavaScript library for producing dynamic, interactive data visualizations in web browsers and is also used for the pie graph on the following page.

Similar to the index page on how the data updates every second, the bar chart also updates every second via the use of Ajax.

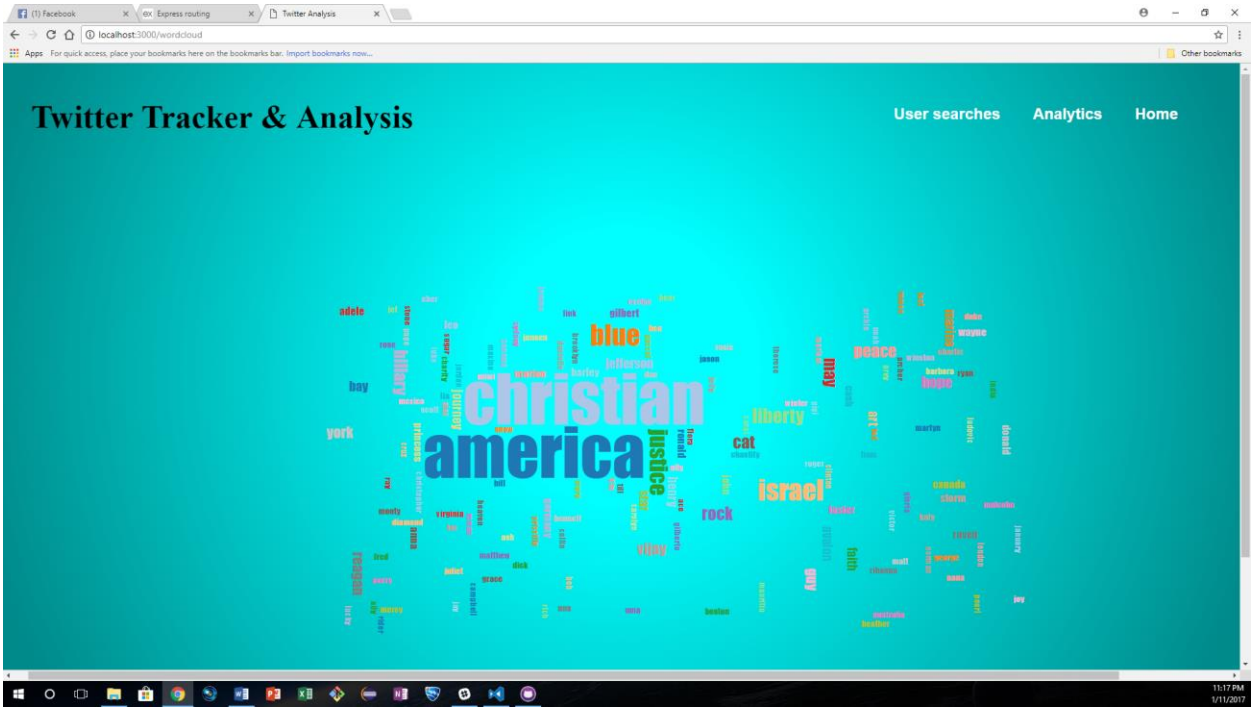
Positive/negative pie chart page



The pie chart shows the ratio between positive, negative and neutral tweets received by the stream. In this case, 43% is neutral, 45% are positive and 12% are negative tweets.

This data was extracted using the sentimental NPM package as explained in the technical description section. The graph itself, like the bar chart was created using D3.js and also updates every second using Ajax.

Cloud word of top 30 words



The cloud word diagram works exactly the same way using the d3 library as per the pie graph and bar chart.

The technical description gives an analysis of how these words are extracted.