Table of contents

Introduction

The project is for analyzing the Twitter big data with Hadoop and map-reduce. There are several important steps for this project. We have taken this big and run our algorithm on it so that we can make sense of it. Our aim has been to collect tweets from Twitter in a large quantity and analyze those tweets for patterns. We have run word counts and K-means clustering on this data trying to infer meaning from it. I will now briefly describe the process we followed.

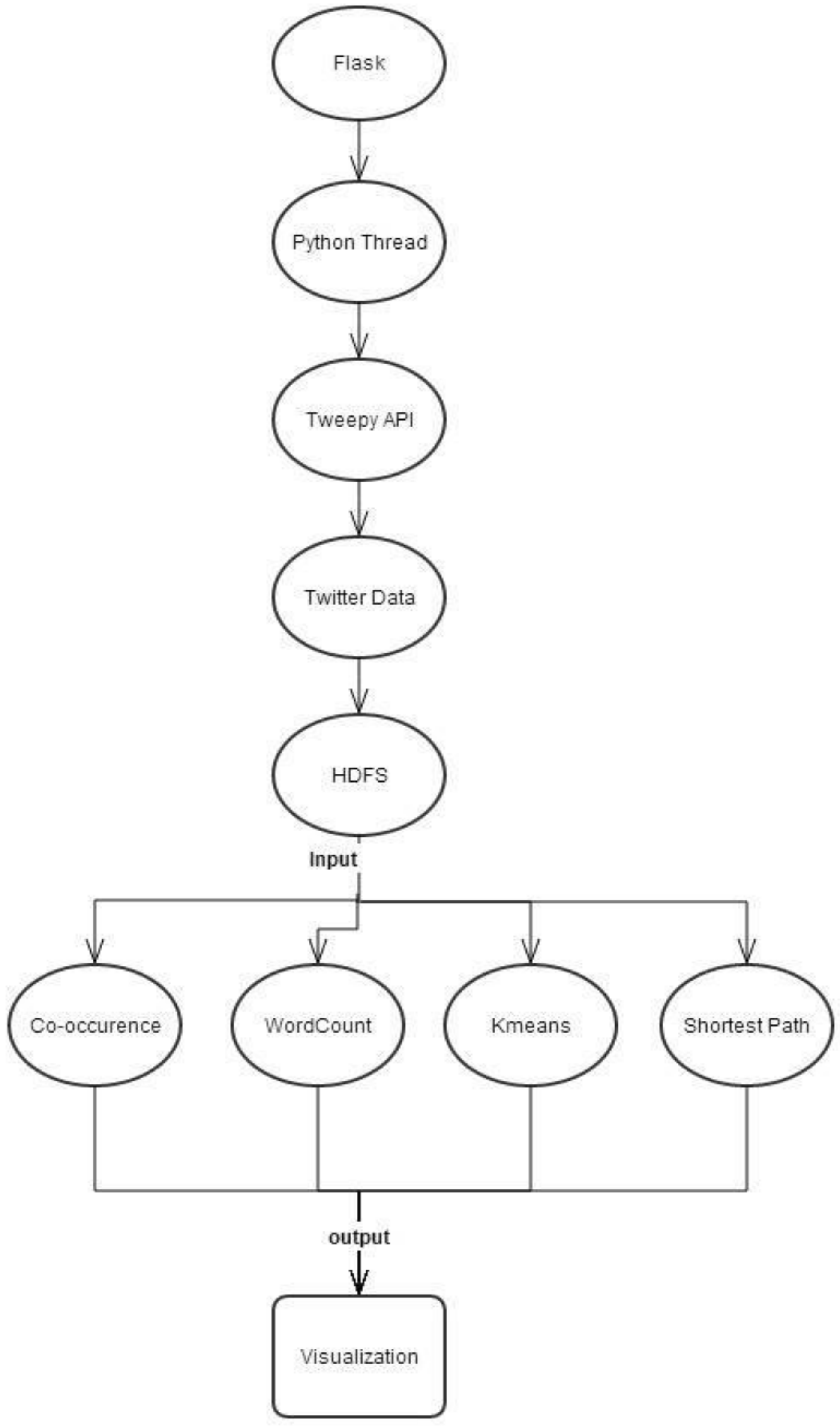
Firstly, data was collected from Twitter. For this we use Python with Tweepy API to collect data, and then we move the data into HDFS where we can get later. We collected around 411,000 tweets at different time intervals on the same day. The data was stored locally and at the end of the script it was moved to HDFS. Where it is available for analysis by map-reduce jobs.

We use java to do the map reduce we believe Java is the best tool to run map reduce jobs. We have made a Maven project in Eclipse with Hadoop 2.2.0 dependencies. We run our map reduce jobs through eclipse assuming input file from Twitter is in place. We split the incoming lines of text, please note each line of text is a tweet, and get the relevant information for the job. In this project, there is four tasks WordCount, Co-occurrence, K-means and Shortest Path. The first three are done on Twitter data and last one on a graph input data provided. Each one is done in a separate job which is initiated through the main class. At the end of each job we sort the output so we can get the most trending words or the co-words with highest frequency.

After do run the four tasks, we get result represented by number and words. We need to do the visualization. We move the output file to the tomcat web app directory where we can access these output files. We use Google Gson API to generate JSON from the output files. We generate JSON for the top ten entries in the reverse sorted file so we can easily get JSON for a particular output file. We then use Java Servlets to send this JSON to the browser front-end.

For the user interface we have made a website which is hosted locally on Apache Tomcat. The website interface for the Data Aggregator as well as the visualization. We initiate the visualization from the webpage at the click of a button on the browser. After we get JSON from Java servlets, we then use Google Charts API to parse this JSON to meaningful visualization. These graphs and charts are shown on the browser.

In the end, we will make some analysis according to the charts. We draw inferences based on the time we got the data and try to figure out what was going on Twitter when we captured the data. Analysis is purely based on the visualization and the output of the map reduce jobs.



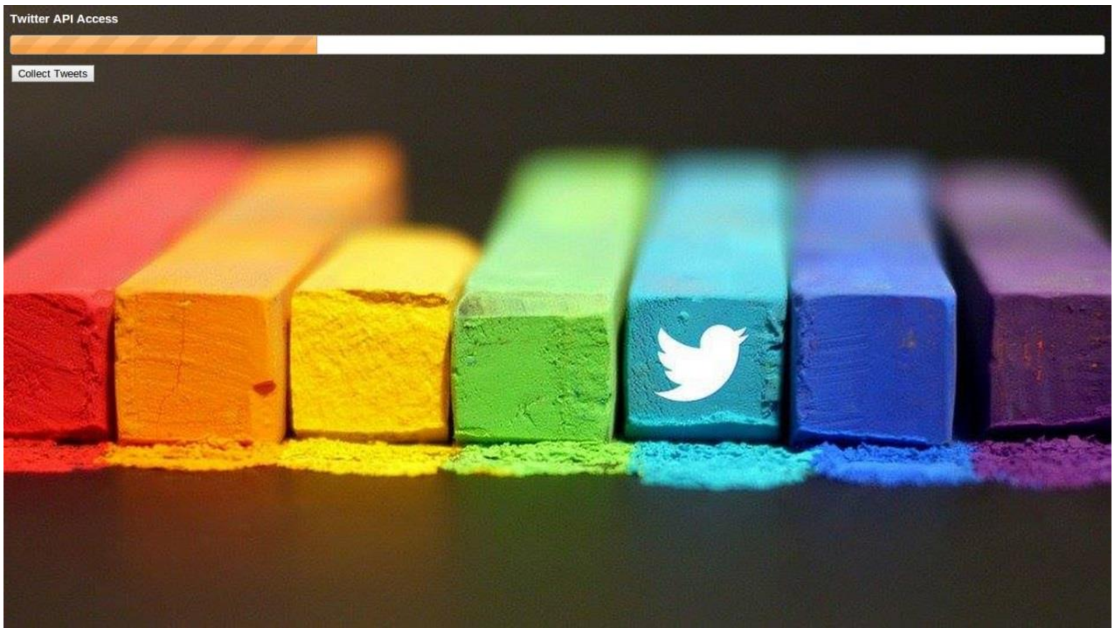
DATA FLOW DIAGRAM

Data Aggregator –Twitter

We have created our own Data Aggregator in Python and designed it from scratch.
The steps for Data collecting is here:

- Data collection is initiated from the browser.
  http://localhost:8080/static/progress.html

- Flask is a python based micro-web framework. Basically it acts as a server to display the front end page of the data aggregator.



- As you can see we can track to progress of our data collection.

- In the backend we have a Python-Flask script running in virtual-env. This script initiates and monitors the data collection.

- The entire Data Aggregator is in Python virtual-env and can be exported and run from any Linux-based machine.

- To start the Data Aggregator you need to run the tweets.sh Bash Shell script from the same directory where Twitter Data Aggregator is located..

- The Python-Flask script starts a new thread every time the button to “Collect Tweets” is clicked.

- We spawn a thread as it the only way to keep track of the ongoing data collection job.

- We then use Tweepy API which a Twitter API for python to get the stream of tweets from Twitter.

- Before we collect tweets, we make one RESTful API call to get the current trends for USA using woeid from Twitter and save them to a file.

- We specify we want only English tweets.

- The Python script collects the tweets locally appending them to file.

- This is the output format for one tweet.

  Username | Follower Count | Tweet Text | #HashTags | @Reply

- First three fields always have data, last two may not contain data depending on the tweet.

- We choose this format after checking if tweet doesn’t have any ‘|’ symbol. If it does we don’t capture that tweet.

- When we reach the counter for number tweets that need to be collected the script stops collecting data and pushes the captured file to HDFS.

- Stats about the data collected.
  i) Number of tweets: 411,561
  ii) Size of File: 55.4 MB
  iii) Trends: 64 trends

- Data was collected over the blood moon eclipse and Jackie Robinson Day. That is 15th April.

Algorithm (Map Reduce)

Counting

- **Word Count**

  Here are the steps for word count algorithm.

  Mapper:
  1. From the input file, split the text from other information.
  2. Use StringTokenizer to break text to tweets.
  3. Send each word to reducer with count of one, if it isn't a common word.

  Reducer:
  1. Get the word.
  2. Add all the number of occurrences.
  3. Output to file.

  Sort:

  Reverse sort to get words with highest count.

- **Trend Count**

  Here are the steps for trend count algorithm.

  Mapper:
  1. From the input file, split the text from other information.
  2. Compare each word of the text with Trends collected from Twitter.
  3. If matched, we send trend with count one.

  Reducer:
  1. Get the trend.
  2. Add all the number of occurrences.
  3. Output to file.

  Sort:

  Reverse sort to get the trend with highest count.

Algorithm remains same for #HashTag count and @Reply count. We sort the output and get the most popular #HashTag and @Reply.

Co-Occurrence

- **Pairs**

Here are the steps for the Pairs algorithm that were followed.
Mapper:

1. From the input file, split the tweet text from other information.
2. Breakup the text to words and add them to an arraylist.
3. Sort the arraylist.
4. Remove any common words.
5. Take one word and pair it with the next and so on.
6. Send each pair to reducer with count of one, if it isn't a common word.

Reducer:

1. Get the pair.
2. Count all the occurrences of the pair.
3. Output the pair.

Relative Frequency:
Run another job for Relative Frequency.

- **Stripes**

Here are the steps for the Stripes algorithm that were followed.
Mapper:

1. From the input file, split the tweet text from the other information.
2. Breakup the text into words and add them to an arraylist.
3. Sort the arraylist.
4. Remove any common words.
5. Take first word check its count and add it as key.
6. Add other words in MapWritable with word as key and count as value, make map value.
7. Send key, value pair.

Reducer:

1. Get key, value pair.
2. Pair key with value in pair and get its count.
3. Output for every key, value pair.

Relative Frequency:
Run another job for Relative Frequency.

K-means

K-means is a way to cluster items, we only need to set the cluster number that we want and run the algorithm. The program will divide all the data into k clusters. The rule that we cluster data can be changed according to our needs.
In this case, Follower Count is our rule to collect our data. Here is steps.

1. We get data from HDFS into map method.
2. In the map method, we split data with follower counts and then we create our key value in the specific format (centroids, follower count) (in this case, we set k=3). And then transfer these value pairs into reduce method.
3. In the reduce method, we need to update the new centroids according to what we have divided. And we will save centroids in the Enum for next time map reduce.
4. We will do iterations between 2 and 3 step, and when the previous centroids equals to the current centroids, we will stop the program in our driver.
5. The output file will be in the HDFS, and we get the data after K-means division.

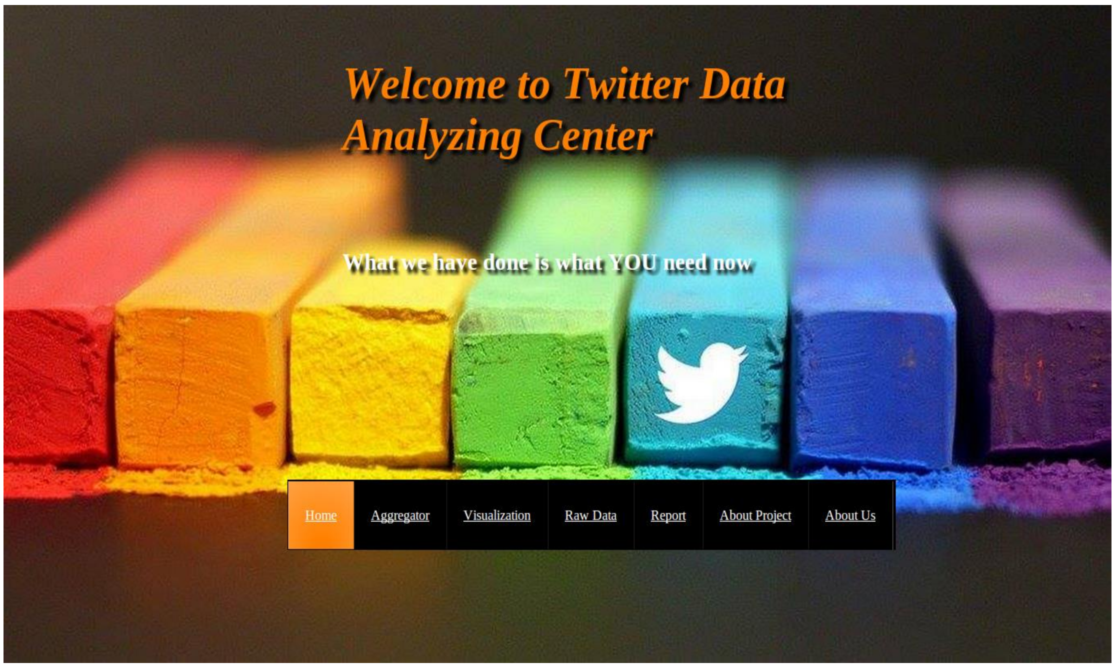
Shortest Path

Shortest Path is using Dijkstra's algorithm to get the shortest path between nodes. Because of the collecting data issue, we use the graph that is given by profession. The format is like this: 1 100 2:3:. The first number is node, and the second number is cost (distance). The 2:3: means that we can go to node 2 and node 3 from node 1 (one direction). The cost is 100+1=101. But maybe that is not the shortest way to get there, so we will do the map reduce to get the shortest path. Here is the steps.

1. We use map method split the data from HDFS, the key is node number and the value the cost. But there is a special case, we need to 100 2:3: as a value and get it into reduce, because if we don't do this, we won't know which node is connected by nodes.
2. In the reduce method, we will find the smallest value with a key. And update it in that special case. And take (key, special format) into the output.
3. One iteration is not enough, we have to use the output of last iteration as the input for the next iteration.
4. When the summation of all nodes cost doesn't change, we will stop the iteration.

Visualization

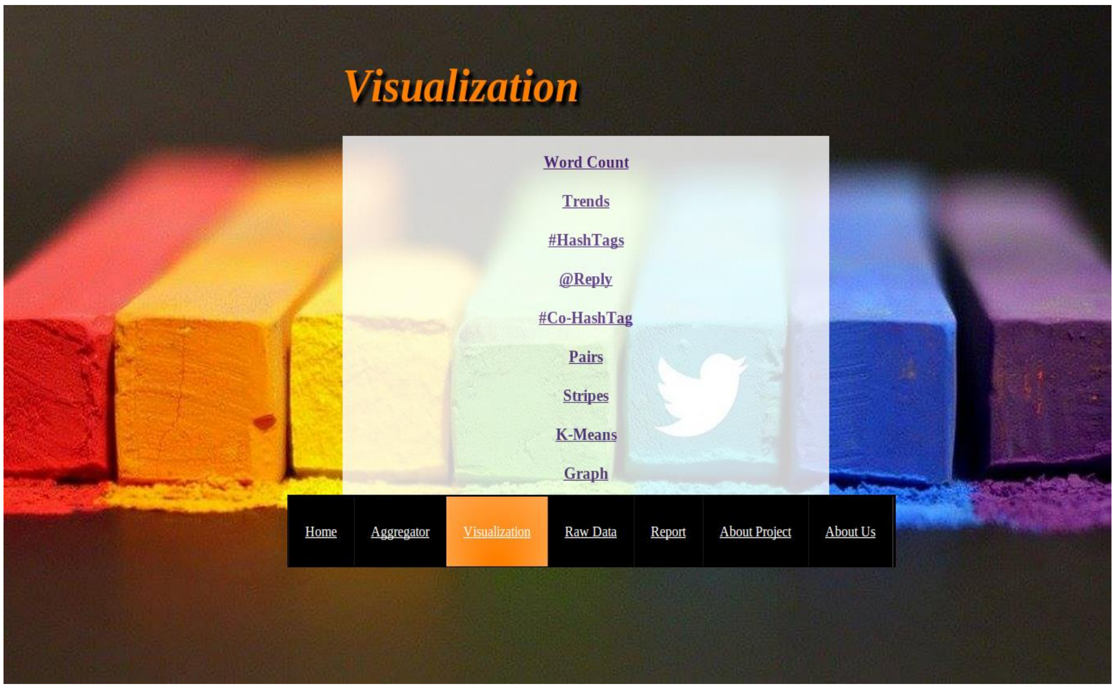
We have used extremely detailed visualization using complex Javascript libraries. As a result we are able to show all our visualizations inside a browser. The visualizations produce typical Javascript effects like mouse over, fade in and clickable pie-charts and bar-plots. For sake of lucidity we have attached all screenshots on different pages. Here is a tour of our website.


Welcome to Twitter Data Analyzing Center
What we have done is what YOU need now
Home
Aggregator
Visualization
Raw Data
Report
About Project
About Us

HOME SCREEN

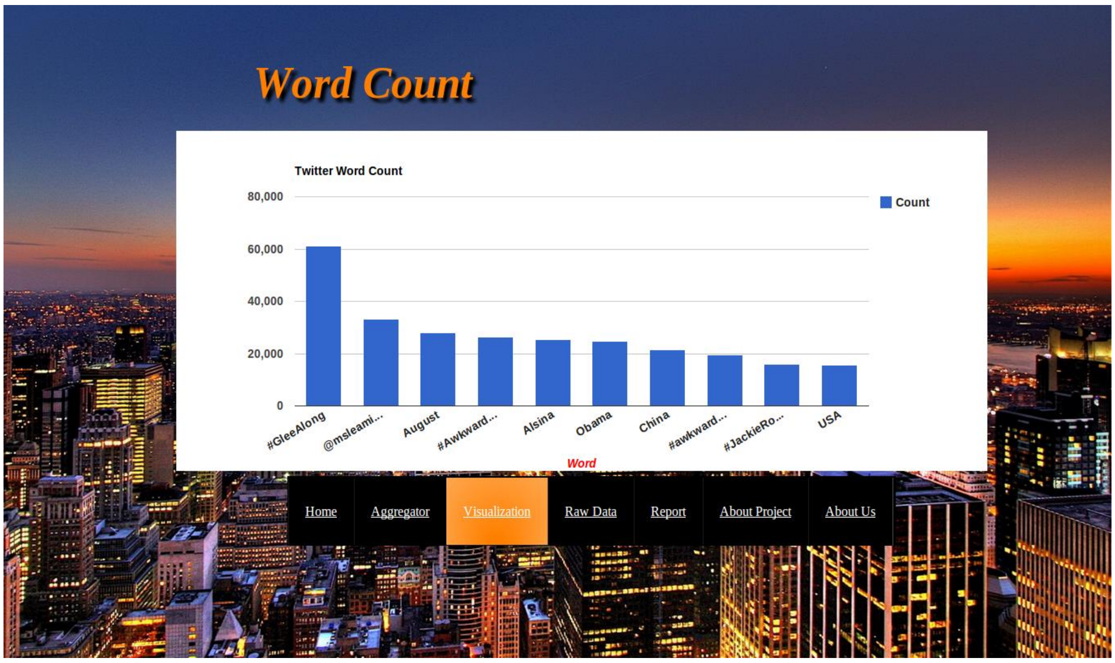
All screenshots are in full screen browser mode taken in Chrome running on Linux. Here is the home page from which you can navigate to any other page on the site.

Default Website address is: http://localhost:8080/Twitter-Project/home.html



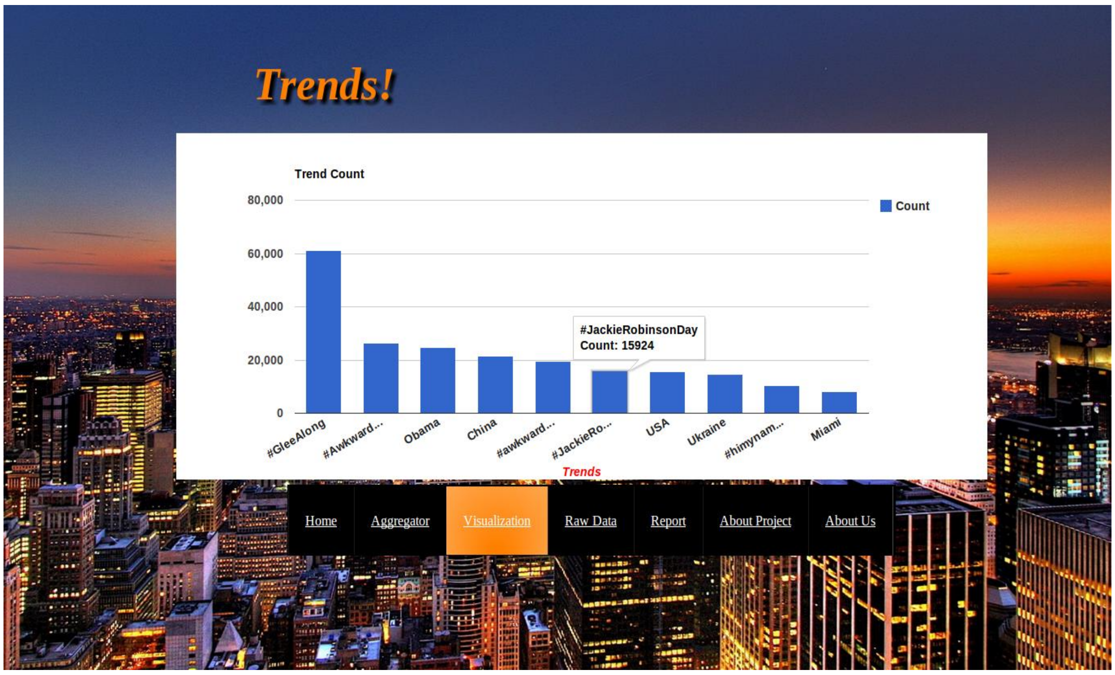
VISUALIZATION SCREEN

This is the main visualization screen. All visualizations are shown here.



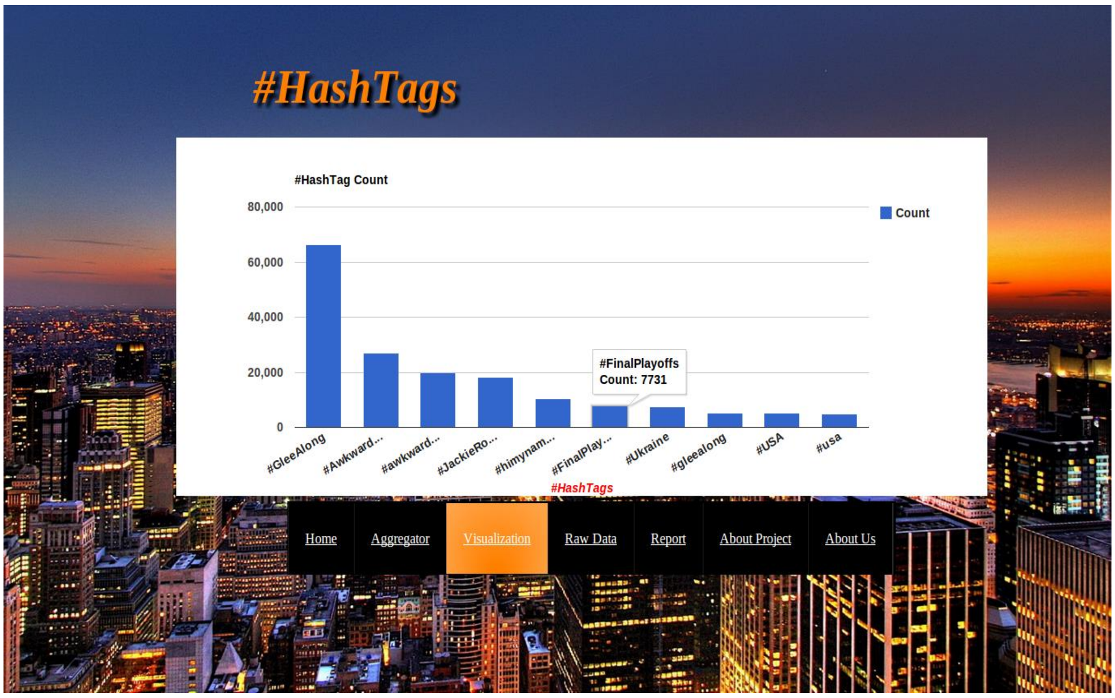
WORD COUNT SCREEN

This is the word count screen. It shows the most popular word is #GleeAlong which is very famous TV show which aired on the same day we captured the data. That is 15th April.



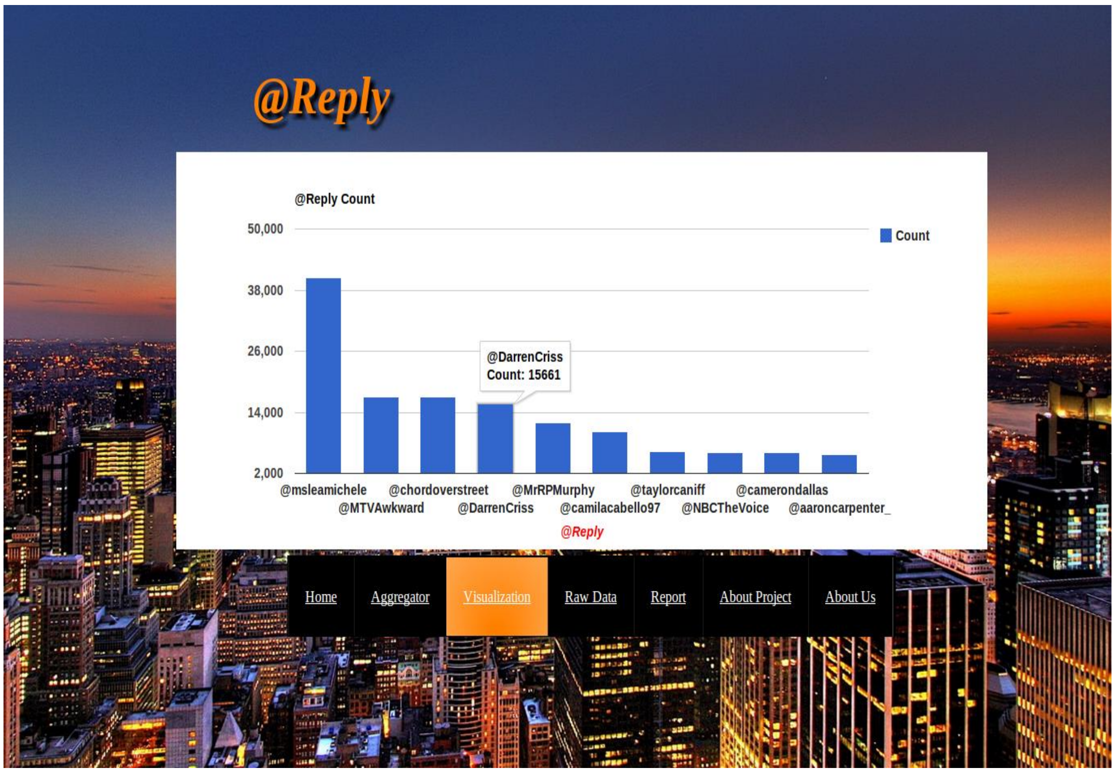
TREND COUNT

This the trend count screen and Javascript mouse over is shown in action. Also, let us remind you that 15th April was Jackie Robinson Day. So, we can see some Twitter activity about Jackie Robinson.



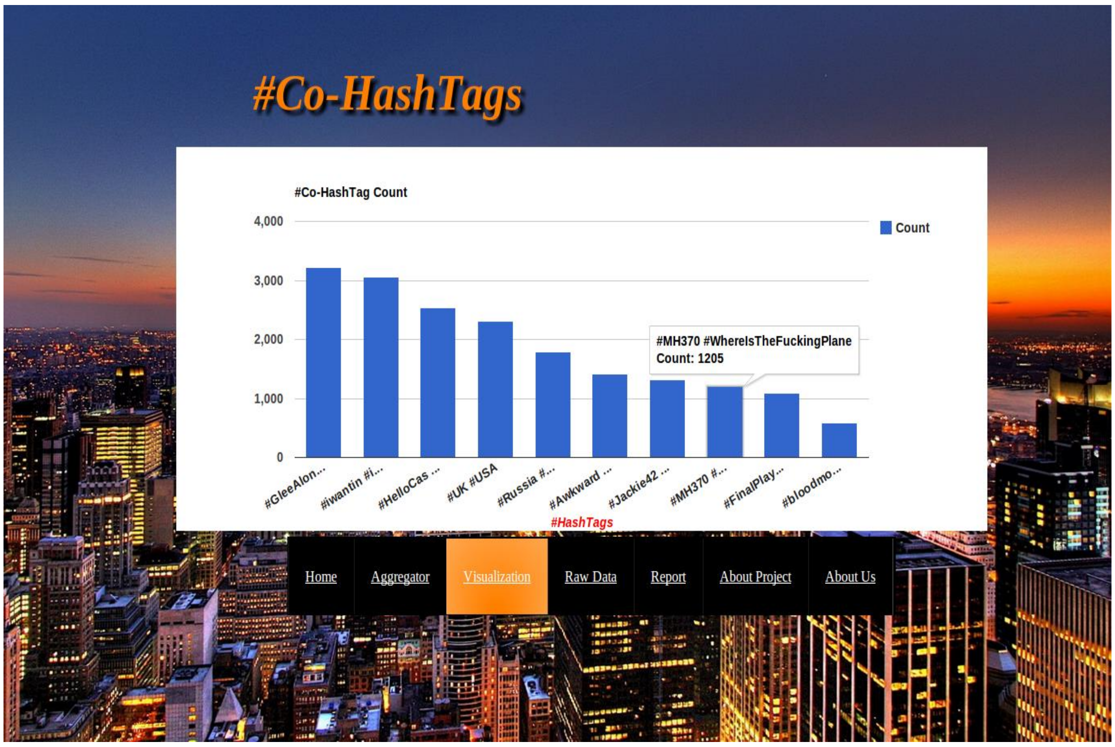
#HashTag Count

This is the HashTag screen and we can see which are the most trending tags at moment when the data was captured. Looks people are looking forward to playoffs.



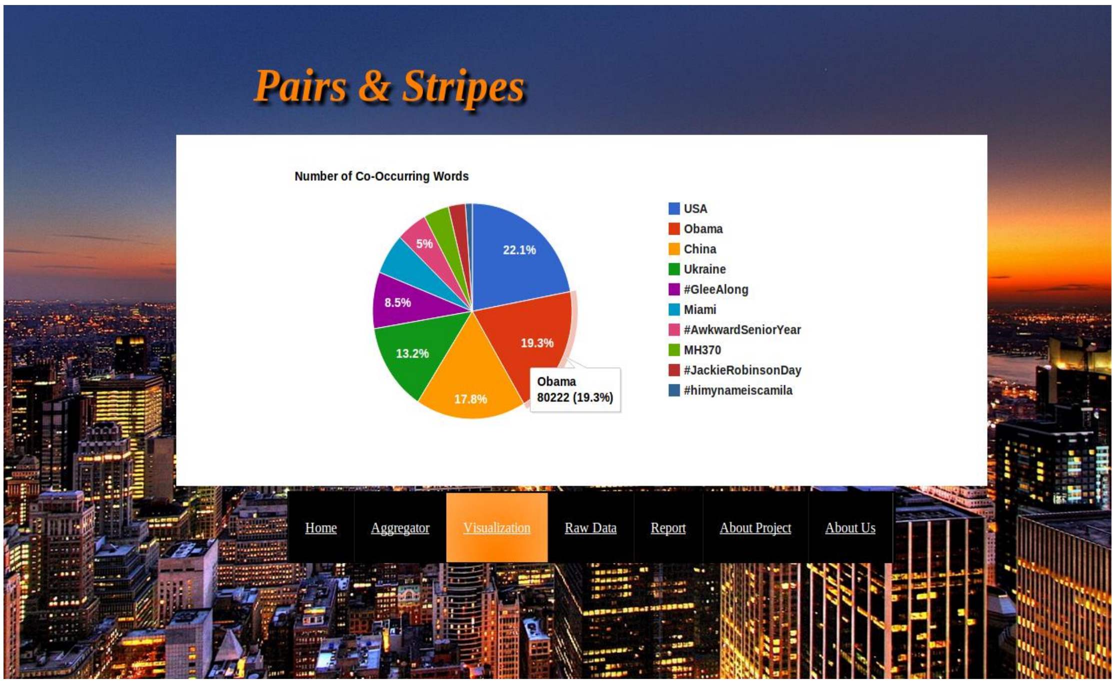
@Reply Count

At the reply screen we see the most reply are to Lea Michele who is an actress on Glee TV show. This seems consistent with our data about the popular Glee episode on the 15th April.



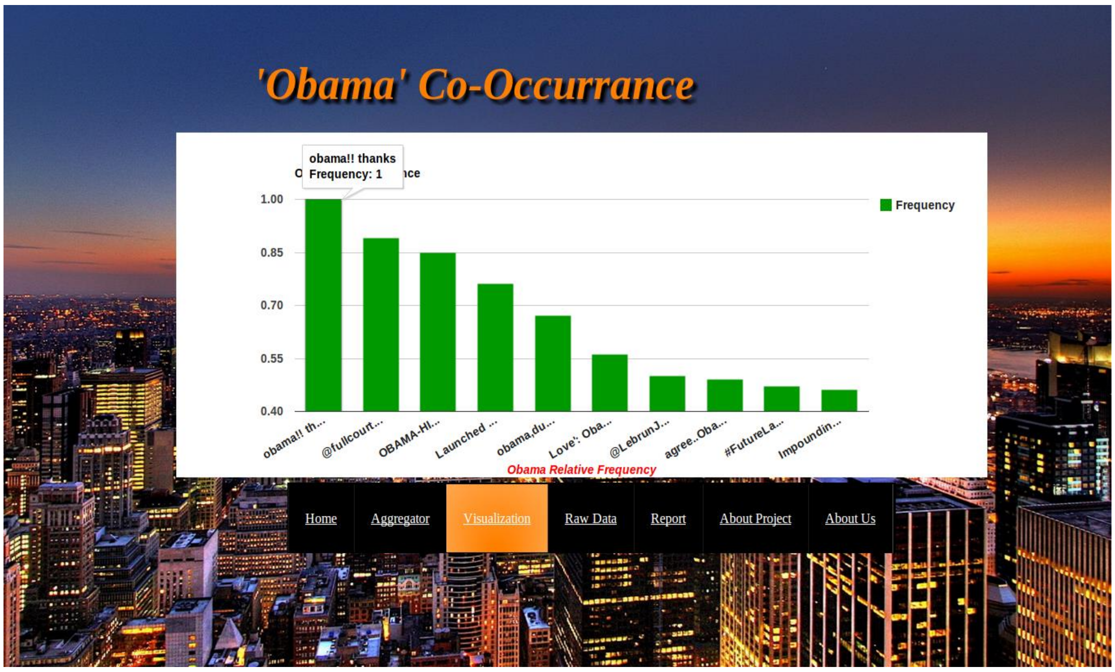
#Co-HashTag Count

As for the Co-Occuring HashTags we can see they still haven't found MH370 airplane.



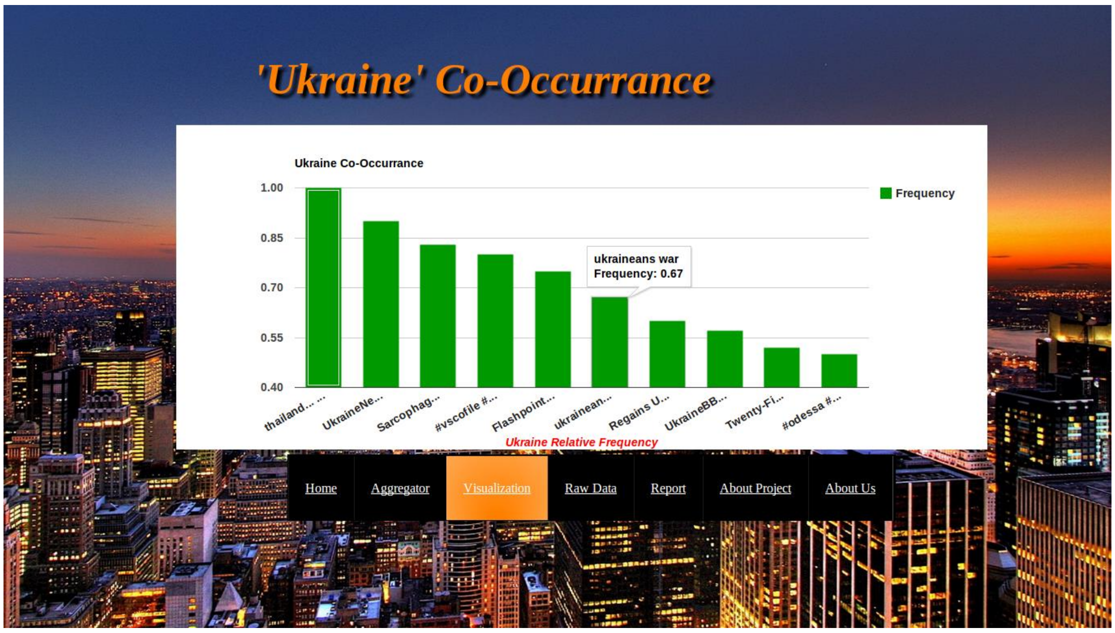
Pairs & Stripes

The pie-charts show the number of unique Co-Occuring words per trending word. So, this means a word in a bigger slice is equivalent to word appearing in long tweets with diverse vocabulary.



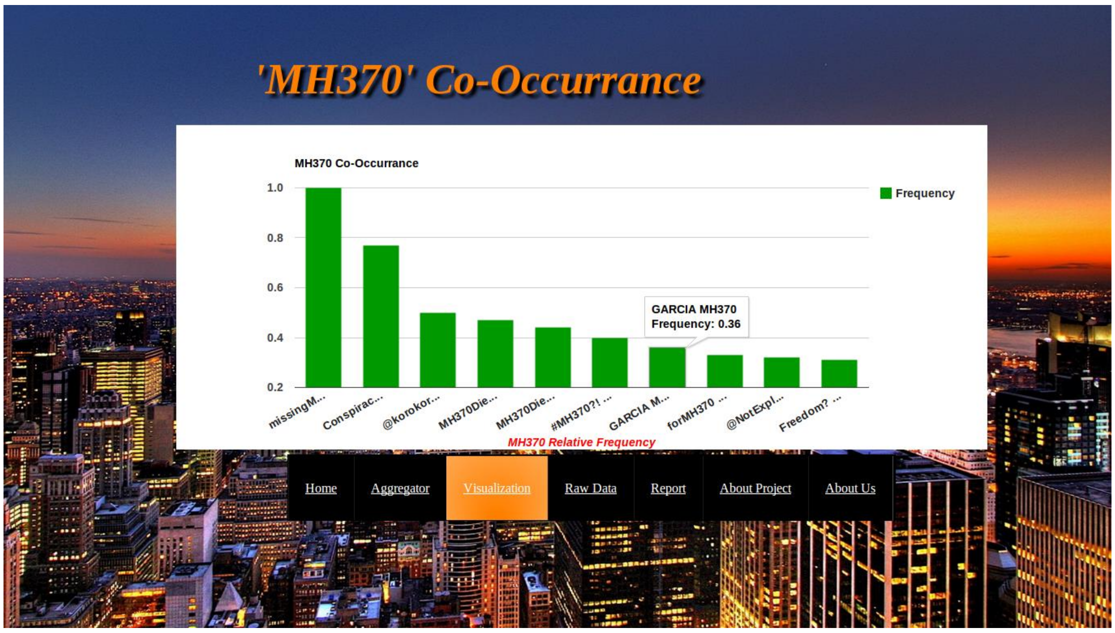
'Obama' Word Frequency

The relative frequency has been rounded off, so 1.0 is ~0.99. We can clearly see Obama is doing well! Well at least when we captured the data.



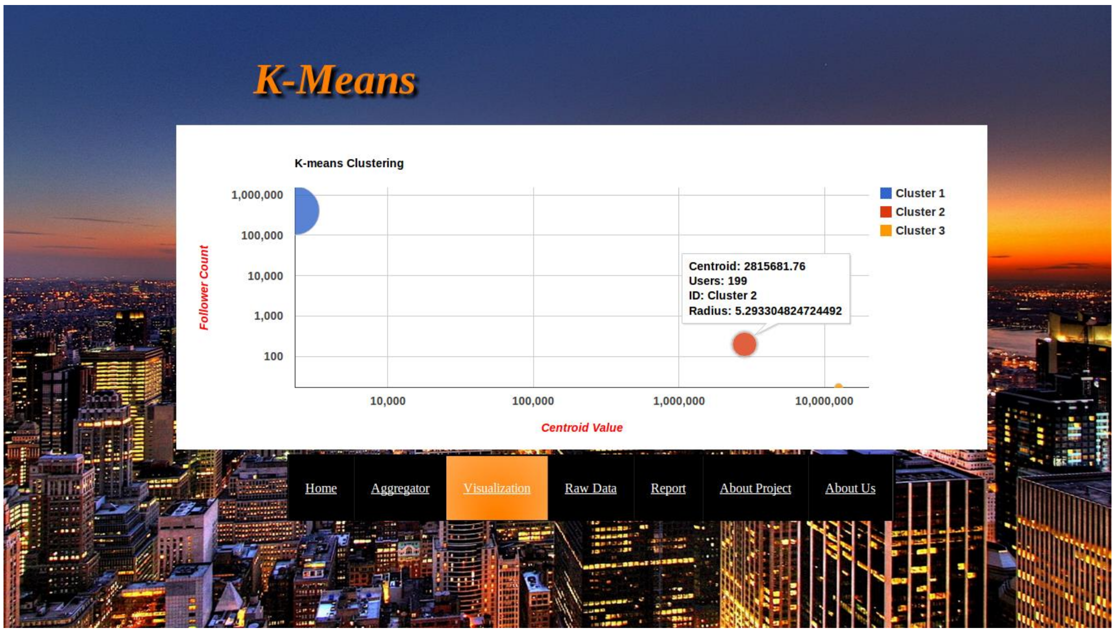
'Ukraine' Word Frequency

Ukraine another word with big vocabulary, long tweets. We can see the conflict in Ukraine is still going on.



'MH370' Word Frequency

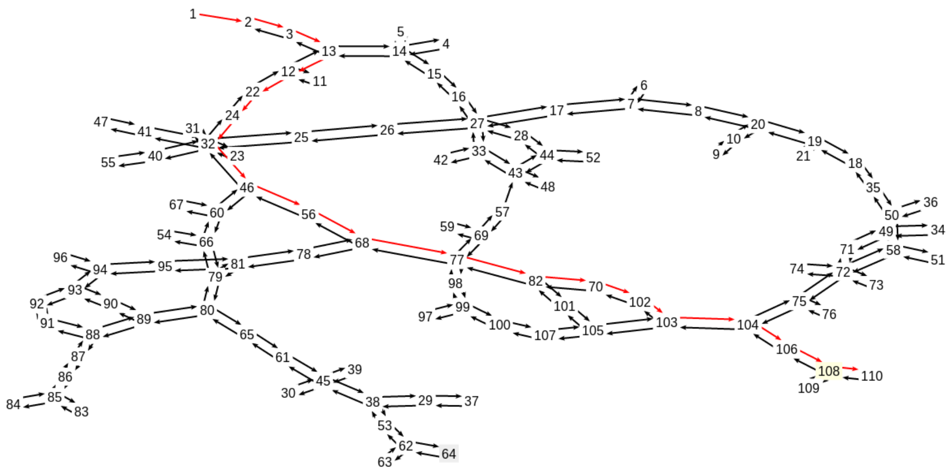
Things looking bleak for MH370. Conspiracy theories are flying about the plane landing in Diego Garcia as show by Twitter data.



Centroid: 2815681.76
Users: 199
ID: Cluster 2
Radius: 5.293304824724492

K-Means Clustering

We get three cluster in K-means. The chart has been log-scaled otherwise the three cluster don't fit in one scale. Bubble sizes show the number of users per cluster. Radius is log of #users. Centroid values are 2,304 and 2,815,681 and 12,000,000 with the first cluster having most users.



Graph Visualization

Javascript masterpiece with moveable graph nodes and edges. Built on-demand with JSON sent from backend. Shortest path is shown in red.

Discovery

Wordcount: We can see some popular words of that day from the figure. The first one is #GleeAlone and second is @misleamichele. They are all from one topic. And then I check it on the internet, the day we collect data is Jackie Robinson Day, it is April 15. There is new episode for the Glee. The misleamichele is actress of the glee. So it is reasonable for these two words in the top. On the 15th, there is new episode for awkward senior year. China, Obama and USA are popular.

Trends: trend is similar with the Wordcount result, but there are some special case because the trend is general word, it can contains all words about the same topic, so there definitely some new trends in the trend list. For example, Ukraine is a hot word for many days because of its political issues. So the trend is reasonable.

HashTag: that is what we got from the twitter data, so the hashtag can show what the key words are in one Twitter data, so hashtag is similar with wordcounts. So the result is almost same with the Wordcount. It is definite indicator that Glee episode was popular.

Reply: Because the misleamichele is in the top of the wordcount, so there is many Twitters about this actress, so it is reasonable that the word 'misleamichele' is on the top. We can see that there are some new words in the reply. 'Camerondallas' is the one. He is very young singer. Maybe there are not contents in the top word count and trends. But it got so much reply, the reason is that maybe there are not some affairs about him, but he is very popular. The number of replies is because of fans attentions.

Co-HashTags: this item is indicated for the two data are in together. Form the wordcount, we can guess that the Glee may be top, when we see the results, we can see that it is what we guess. There are some double words together, just like Ukraine and Russia, and also there are some words that are almost same, just because of a space like 'I wan tin' and 'Iwan tin'.

Pairs & Stripes: The pie-chart shows number of unique pairs per popular trend. This means that the number of different pairs for political tweets on Topics like Obama, China or Ukraine are longer and have diverse vocabulary. Glee has a higher wordcount but its tweets are smaller and repetitive. Stripe computes the same in a more efficient way.

i) Long tweets with big vocabulary. (Smaller Word Count)
   a) Obama
   b) China
   c) Ukraine
   d) USA

ii) Small tweets with repetitive vocabulary. (Larger Word Count)
   a) Glee (TV show)
   b) Awkward Senior Year (TV show)

K-means: From the figure, we can see that the follower counts can be divided into 3 clusters. For the most of users, they most are in the 2,304 centroid cluster. Ordinary users have a smaller word count. Most of Twitter users are your ordinary users so the bubble is biggest even after log-scaling it. Second bubble is your TV stars, celebrities or popular brands or organizations. Popular people and popular content have around 2 million followers but are lesser in number, so smaller bubble. Smallest bubble of people are worldwide known companies, Hollywood movie stars with a huge fan following of around 12 million followers. Of course you can count these people on your finger-tips, so that is your smallest bubble.

Conclusion

“If, most of the tweets are dumb then so are most people on Twitter.”

If we are to conduct the same analysis with new Twitter dataset I’m sure the result will be the same. How is this useful? Twitter is a great place for advertising and trends on Twitter change very quickly. This can be leveraged by business to catch on the latest fad and advertise something similar. Let’s say Demi Lovato a famous singer is getting lot of replies on Twitter. It would be great time to advertise the brand of lipstick she endorses.

Of course, these are my own conclusions and entirely contestable. But this was a great project and I learned lot of new technologies. I will list all of them here:

1. Hadoop 2.2.0
2. Map Reduce
3. Python Virtual-Env
4. Flask
5. HTML
6. JQuery
7. Javascript
8. Java
9. Java Servlets
10. Maven
11. Eclipse
12. Google Gson API
13. Google Charts API
14. Spring.JS