CSE4867/587 Project Report Details

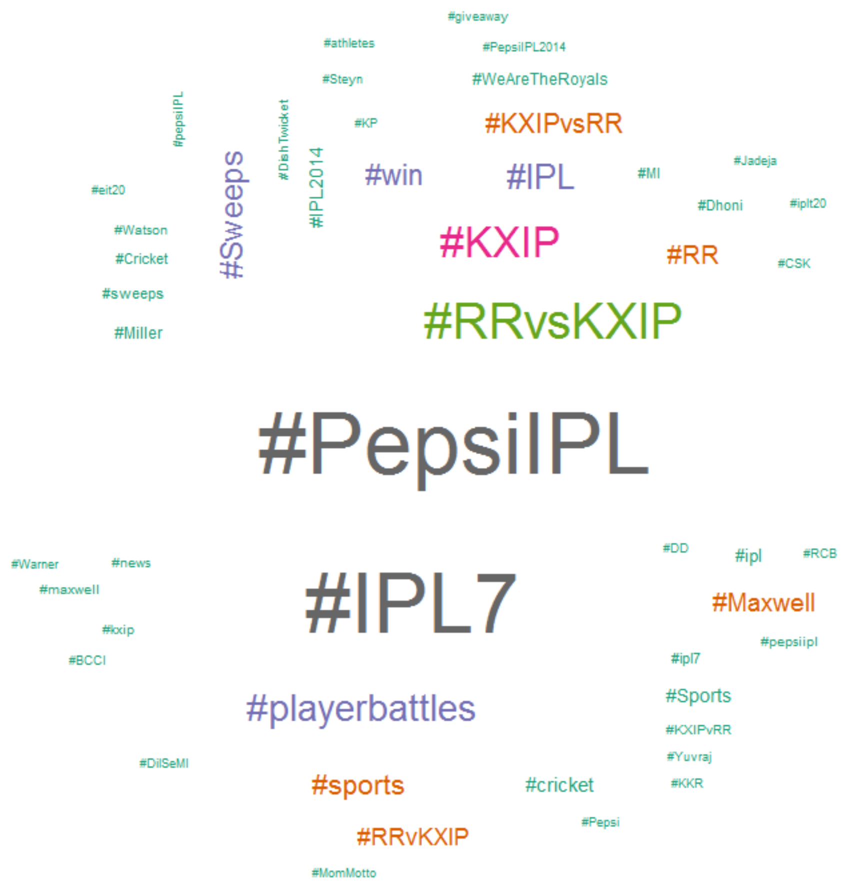
BIG-DATA CONTENT RETRIEVAL, STORAGE AND ANALYSIS FOUNDATIONS OF DATA-INTENSIVE COMPUTING

Author:
Mayank Agarwal
Siddharth Srivastava

Masters in Computer Science
University at Buffalo

Submitted By
Mayank Agarwal(magarwal@buffalo.edu)
Siddharth Srivastava(srivasta@buffalo.edu)

Table of Contents



1. <u>Abstract</u>:

This project is aimed towards learning and understanding the basic concepts behind Hadoop Architecture. We also learn the implementation of Map Reduce to do basic jobs like word count, co-occurrence, K-Means, and shortest path. We come up with multiple inferences to the data using analysis done on data produced by MR.

One of the objectives of this project is to collect data using Twitter Streaming API. We analyze twitter hashtags, user data, follower count and tweet text that we aggregate.

We analyze large datasets using R. We analyze the meaningful data using the analysis done on different column attributes by plotting numerous graphs like, Histograms, Pie-Charts, Box plots, Bar plots, maps and other graphs.

The main plan behind it was to learn different concepts in Map Reduce, HDFS. We learnt concepts like Counters, Secondary Sort, Custom Partitioner and iterations on Map reduce.

While plotting and mapping, we came across many cool and beautiful things. We came across "GEPHI", tool to create a graph using edges of the network. We also learnt the concept of WordCloud, which creates a cloud of words placed logically and its size depending upon the frequency of that word.(in our case it is the hashtags).

You can check out more on http://aegissid.wix.com/dicproj

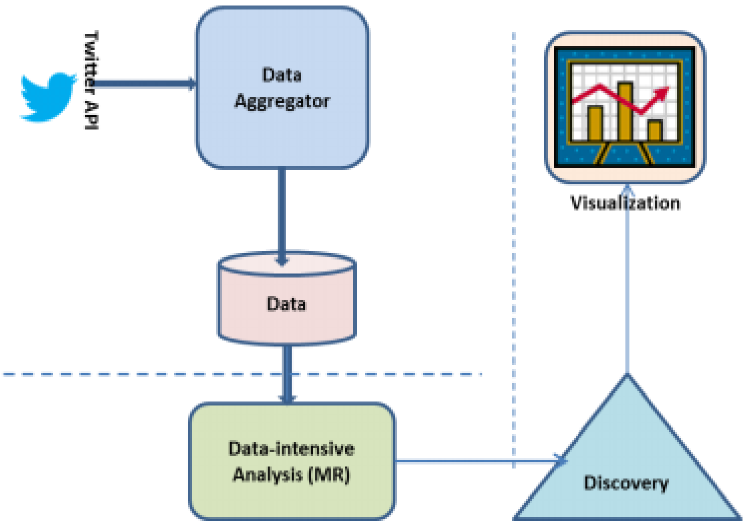
2. Project objectives:

We are going to perform MR on aggregated data and at the end of the project, the following objectives will be covered:

- To understand the components and core technologies related to content retrieval, storage and data-intensive computing analysis.
- To understand twitter API to collect streaming tweets.
- To explore designing and implementing data-intensive (Big-data) computing solutions using Map Reduce (MR) programming model.
- Setup Hadoop 2 for HDFS and MR infrastructure.
- Thorough analysis of data using R and GEPHI.
- To visualize the data using appropriate tools.
- Conclude meaningful knowledge from analysis.

3. Project Approach:

1. We started with the basics and went through the theoretical knowledge of handling large datasets and data science. We learnt techniques and different ways to handle Big Data.
2. We went through the basics of Map Reduce and Hadoop. We tried out basic programs and then used the same to build upon the other analysis.
3. In the meantime, we also started collecting data using Twitter streaming API. More details will be explained in the diagram below.



System Architecture for Data-intensive Analyzer

- **Twitter API:** The Streaming API is the real-time sample of the Twitter Firehose. This API is to collect large data related to tweets, hashtags, users, locations, followers and retweet count. We can use the API to filter out specific tweets based on hash tags.

- **Data aggregator:** Once the data is collected from Twitter API, we clean the data from unwanted words, like stop words, special characters and then arrange it in a customized order to properly analyze it.

- **Data:** After getting the data in the required format, we copy it into HDFS, so that we can run map reduce on it.

- **Data-Intensive Analysis(MR):**
    - Designing and implementing the various MR workflows to extract various information from the data. (I) simple word count (ii) trends (iii) #tag counts (iv)@xyz counts etc.
    - Implementing word-co-occurrence algorithm, both "pairs" and "stripes" approach.
    - Clustering: using Map Reduce version of K-means discussed in class.
    - Applying the MR version of shortest path algorithm to label the edges of the network/graph.
- **Discovery:** The final output of the MR gives us specific hashtag counts, co-occurring tags, follower counts and tweet word count. This meaningful data is then converted into other formats to be used by R and GEPHI.
- **Visualization:** We plot different graphs, bar charts, Histograms, pie charts and other maps to give the data some meaning. Using these visualizations, we give some meaning to the data.

4. Installing Hadoop 2.2.0

- **Installing guide URL**: *https://drive.google.com/file/d/0BweVwq32koypbjd6T19QWmhUZlU/edit?usp=sharing*

  We followed the installing guide to install Hadoop on our machine
  Virtual Machine configuration:
  - Ubuntu 12.04.4 x64
  - 2 CPUs, 2GB RAM
  - 12GB Virtual Hard Disk

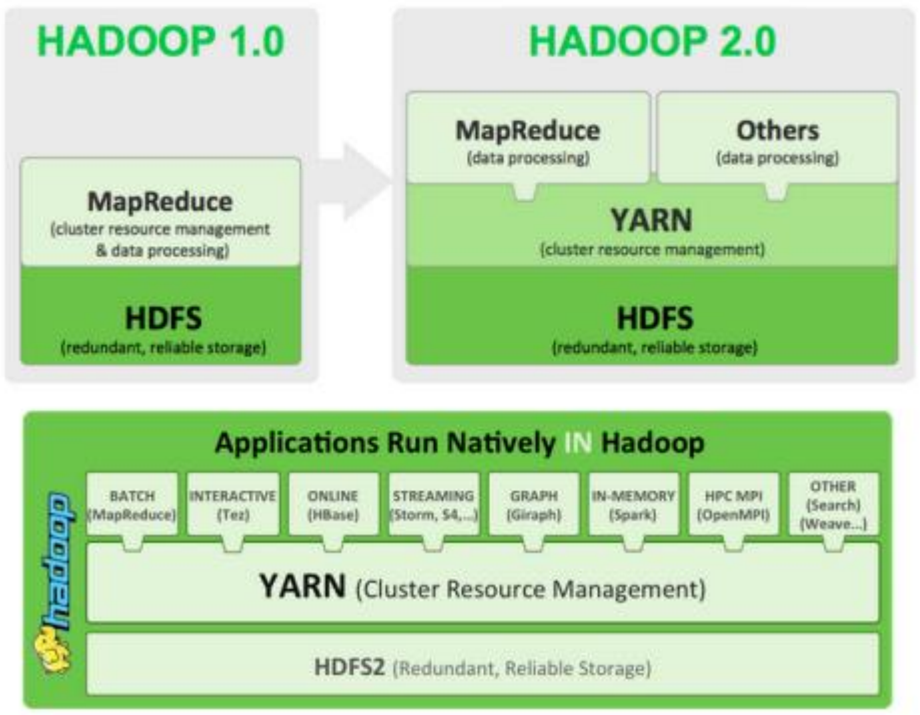

HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)
Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

Fig: Hadoop Infrastructure

The setup enabled us to understand the concepts in depth :

- HDFS
- Yarn
- Node Manager
- Resource Manager

During the installation, we came across some problems, but after some trouble shooting, we solved it, it helped us understand the architecture better.

5. <u>Phases of Data Collection (Twitter Streaming API):</u>

<u>5.1 Twitter Data collector</u>

We used Twitter streaming API to collect twitter data. We were already provided with this API. We collected specific tweets only, giving IPL and Election related filter hash tags.
We collected the data for around 8-10 days and collected large data. We collected the following data:

- Tweet text
- Follower count
- Date
- User name

<u>5.2 Data Cleaning Process</u>

We had to clean the data as we had a lot of unwanted words and characters. We got rid of stop words, commas, colons and other punctuation marks. Raw data gave absurd counts.

<u>5.3 Reformat data</u>

After getting the required data, we had to format it according to our requirement. We placed proper delimiters so that the mappers in MR would understand the data. We created our own custom Parser to create this data.

Eg. of a tweet:
RT @BenWaring98 Good day training today @harry_jerome @Georgekenny96 #cricket , Mon_Apr_21_15:03:24_EDT_2014 , 0 , Harry_ , 599

<u>Total Tweets :</u> ~550,000

Check out more details on http://aegissid.wix.com/dicproj

6. Data-Intensive Analysis of Twitter Data using Map Reduce:

6.1 Implementing word count for “HashTags”, “@xyz”, and Tweet text

After collecting the data and formatting it in the required format, we run Map Reduce on it to count the number of Hash tags, @users and words. So we get the trending words.

We implemented a custom partitioner to create different files for hash tags, @xyz and other words.

Algorithm:

Driver Class –

1. Initialize job.
2. Set attributes like mapper class, reducer class, output keys and values, number of reduced tasks and partitioner.
3. Run mapper
4. Run partitioner
5. Run reducer

Mapper Class –

1. Tokenize the line.
2. Analyze each token.
3. Emit(token,1)

Partitioner Class –

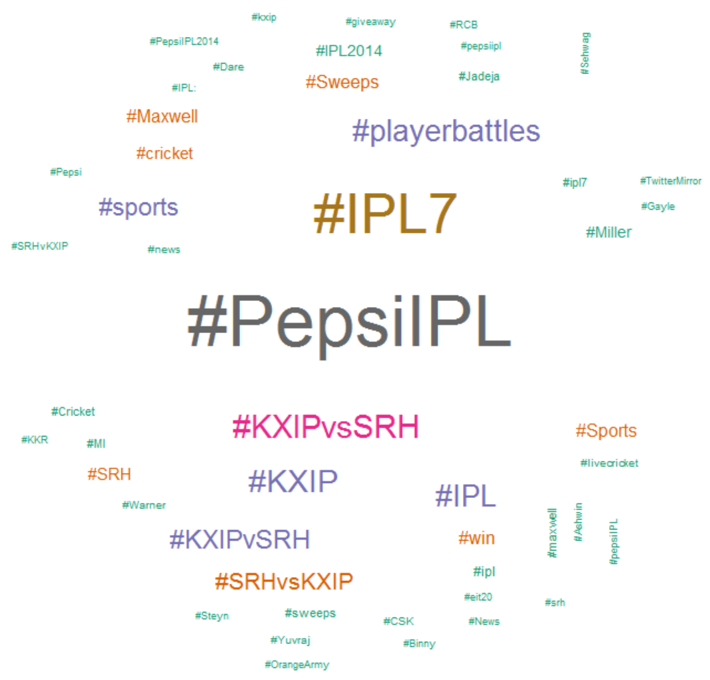
1. Check if token starts with #.
    a. Return part1
2. IF token starts with @.
    a. Return part2
3. Else
    a. Return part0.

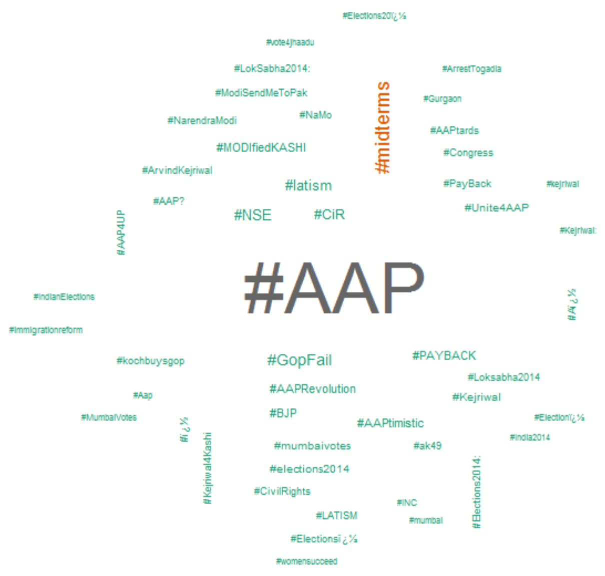
Reducer Class –

1. Assign sum = 0
2. For all values of a token
    a. Sum = sum + value.
3. Emit(key,sum)

6.1.1 Word count implementation

We get the most trending words and we can see that in the graphs.



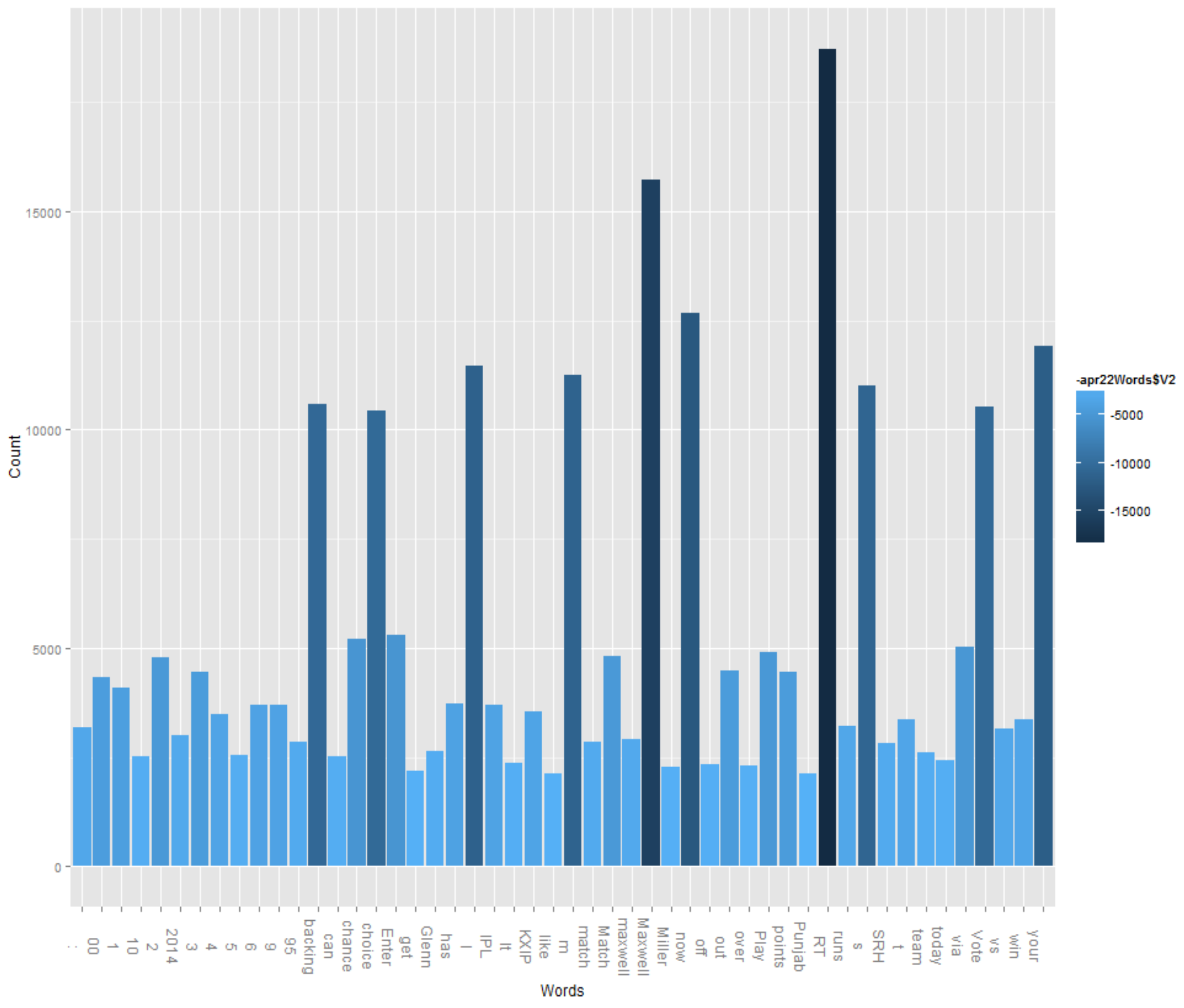


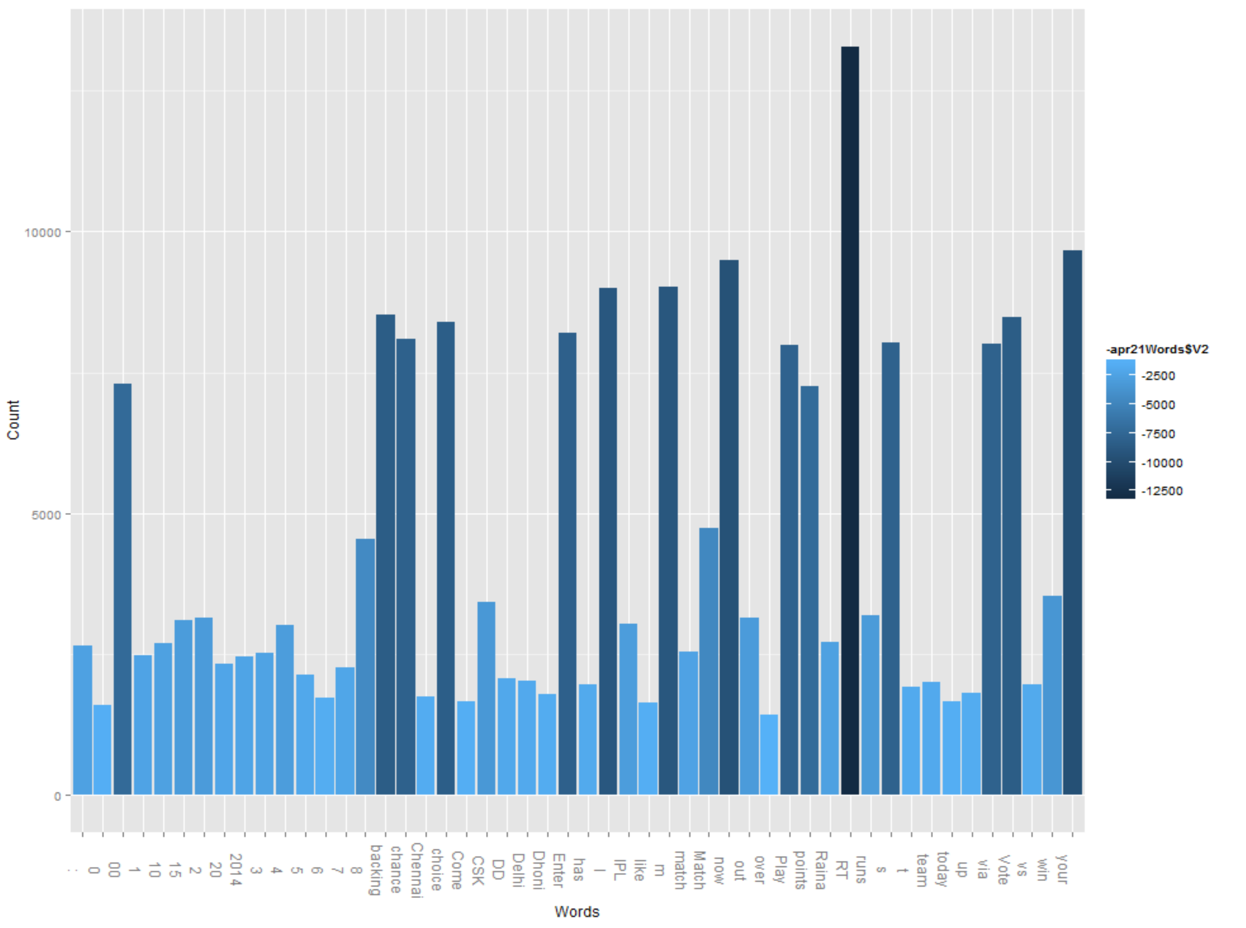
Here we can see that before IPL had started the tweets were all about elections and market. We can also make out that people were more after AAP.
After IPL started, we can see there are no tweets related to elections any more. It is all about teams and cricket. This shows how quickly people change their mind.

<u>6.1.2 Most Trending words</u>

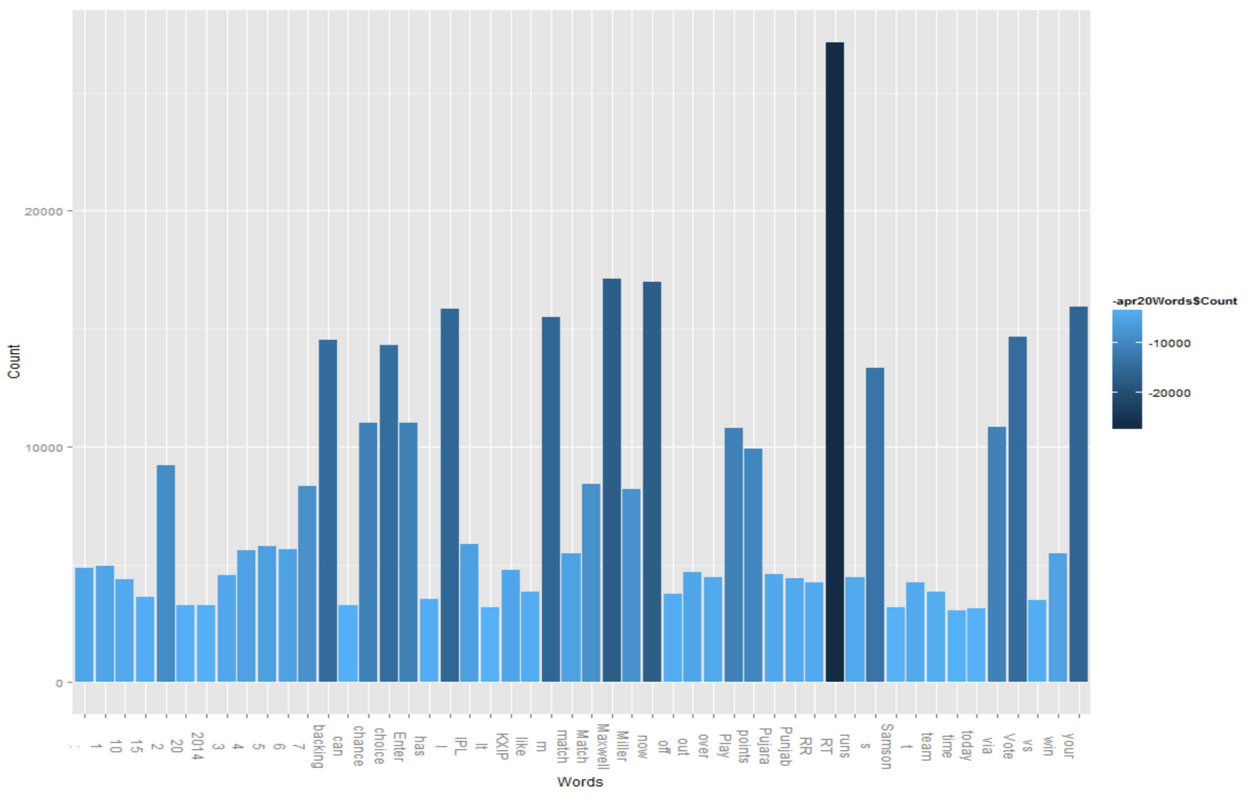
We have sorted the incoming word count output using R.
Command to sort :

apr20Add=apr20Add[order(-apr20Add$V2),]

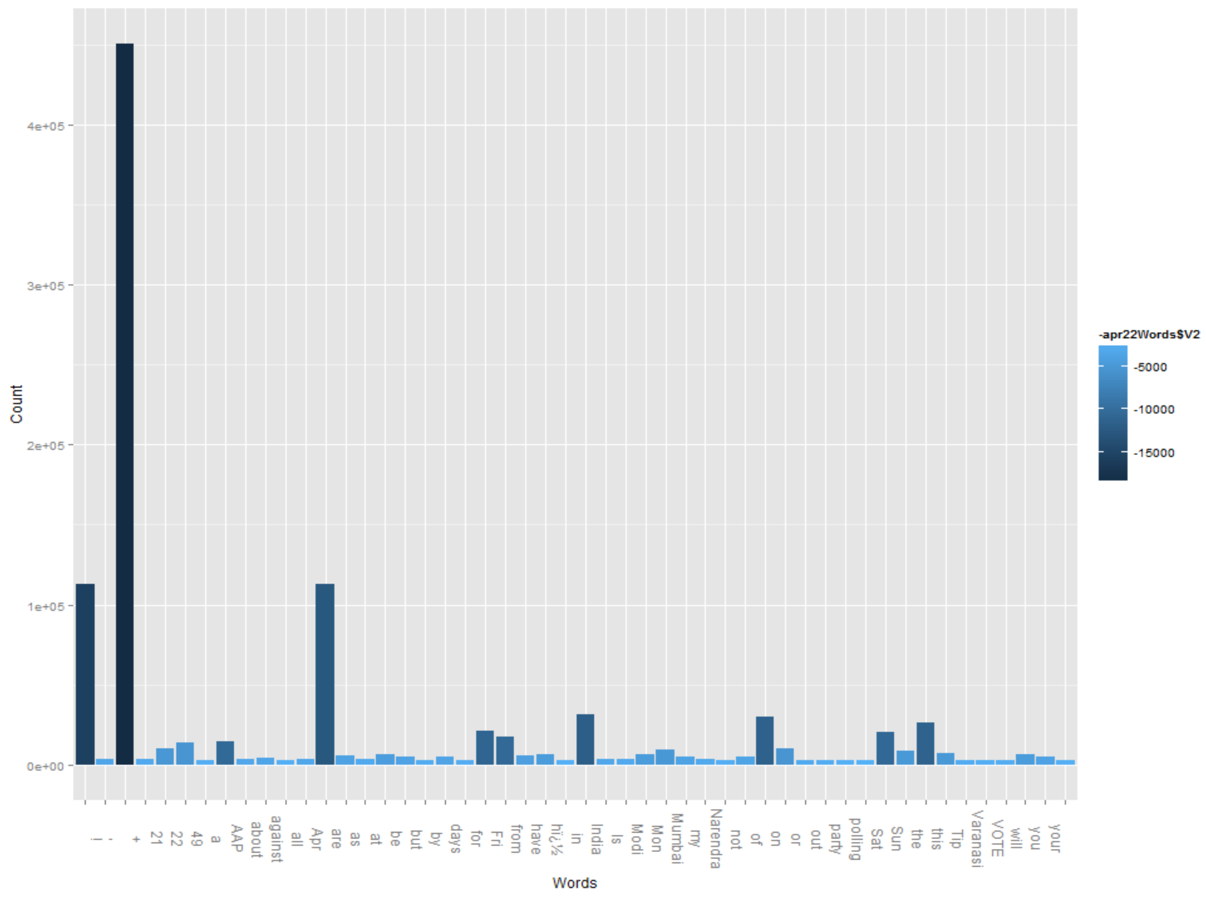




Count
Words
-apr21Words$V2
-2500
-5000
-7500
-10000
-12500
0
5000
10000
: 0 00 1 10 15 2 20 2014 3 4 5 6 7 8 backing chance Chennai choice Come CSK DD Delhi Dhoni Enter has I IPL like m match Match now out over Play points Raina RT runs s t team today up via Vote vs win your



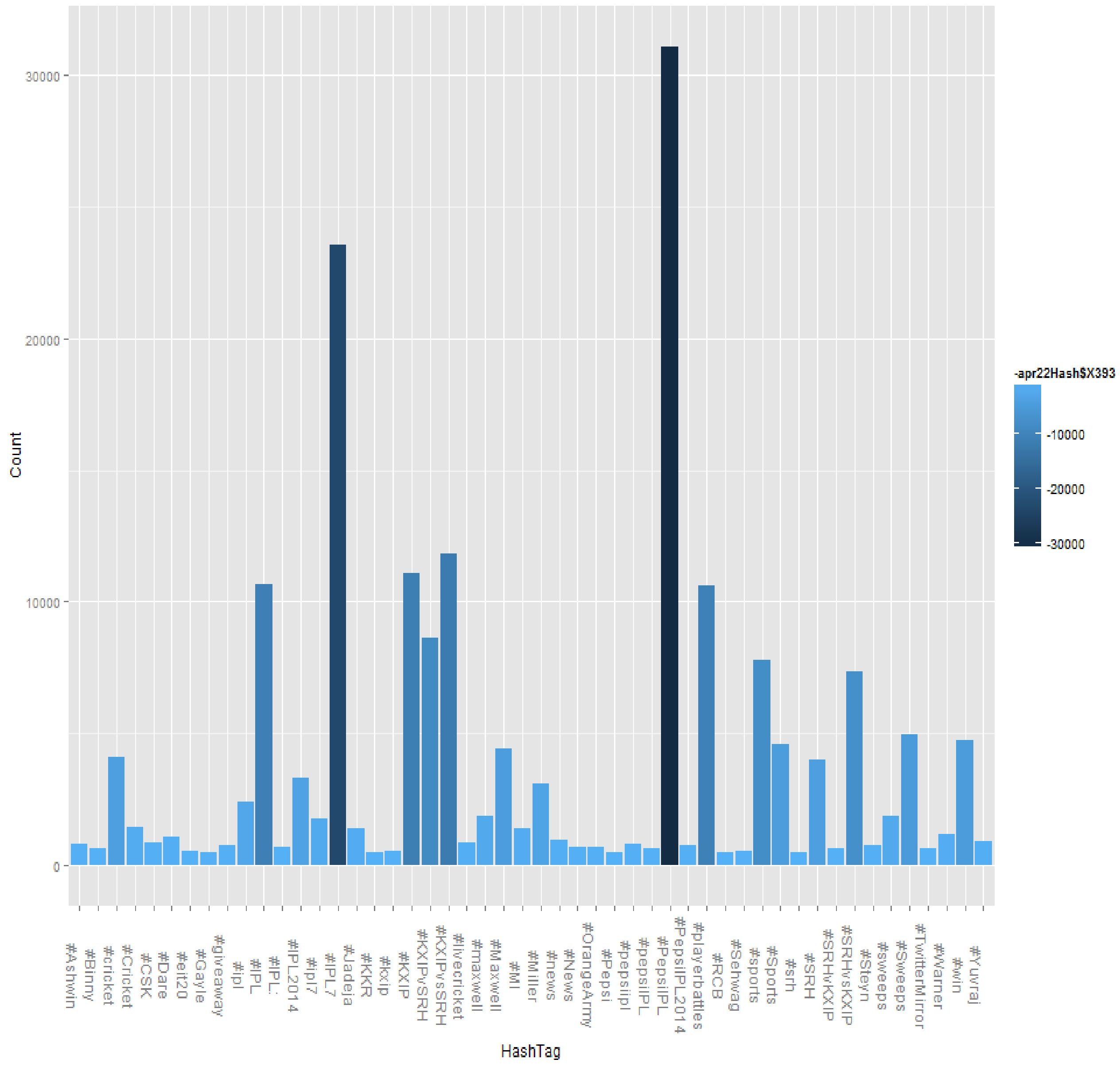
Count
Words
-apr20Words$Count
-10000
-20000
0
10000
20000
: 1 10 15 2 20 2014 3 4 5 6 7 backing can chance choice Enter has I IPL it KXIP like m match Match Maxwell Miller now off out over Play points Pujara Punjab RR RT runs s Samson t team time today via Vote vs win your

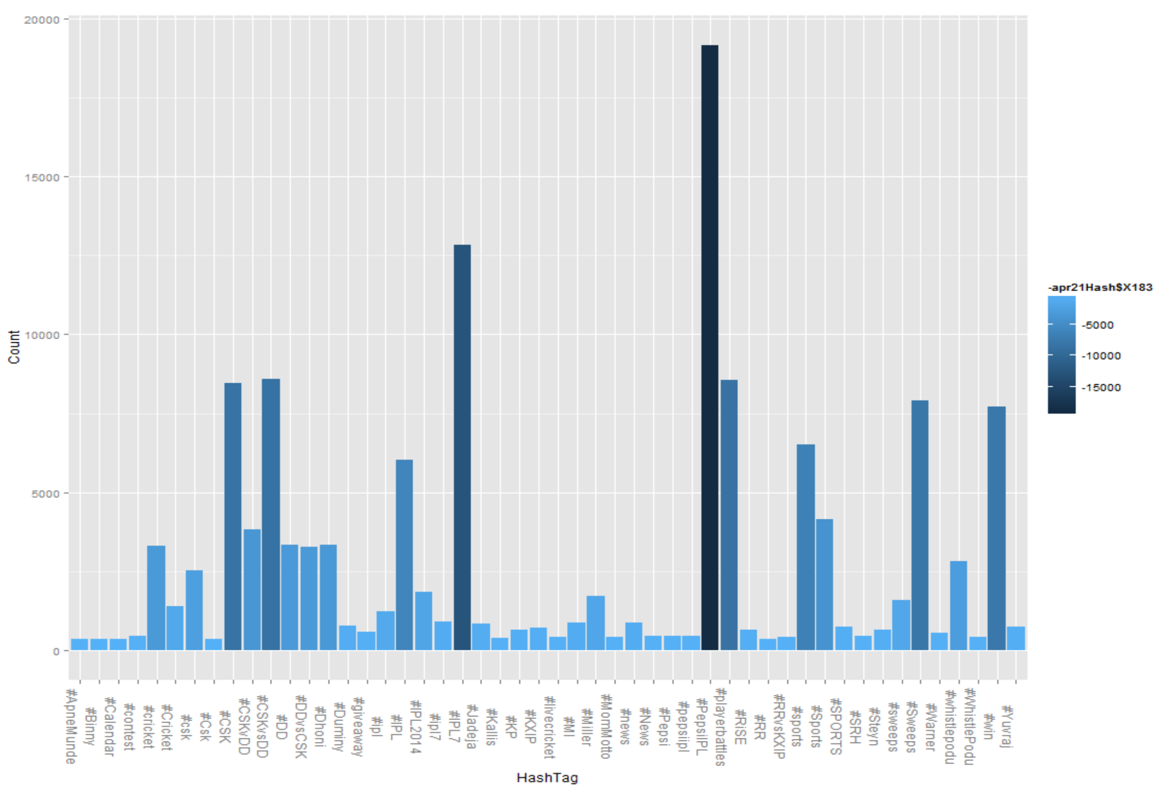


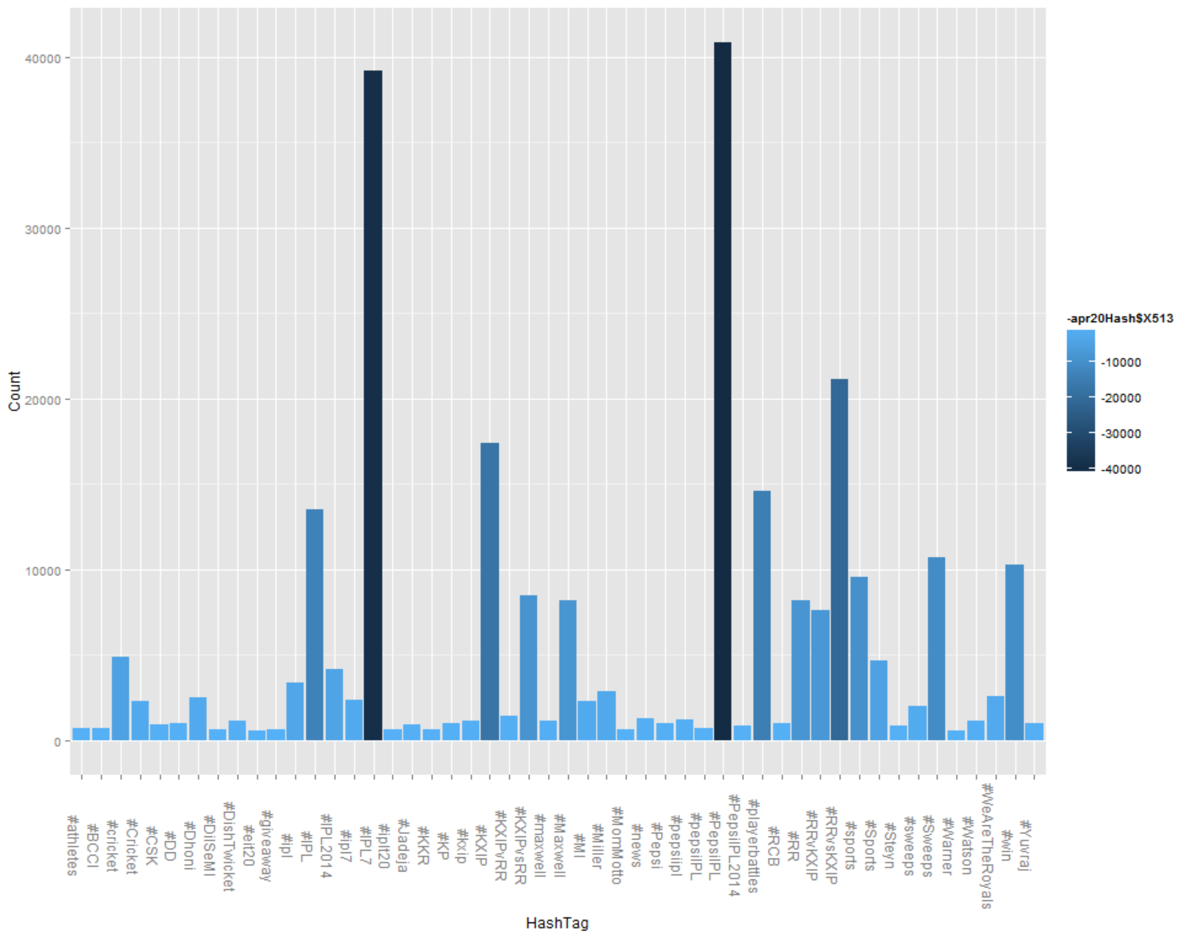
Seeing this data, the data has been sorted according to the count and in alphabetical order. We can see that most of the tweets were for IPL as the IPL started. A lot of the tweets came for Dhoni, Yuvi, Kohli, which shows their popularity.
Before IPL started, it was all on elections and we can see that major tweets were related to AAP and Narendra Modi.

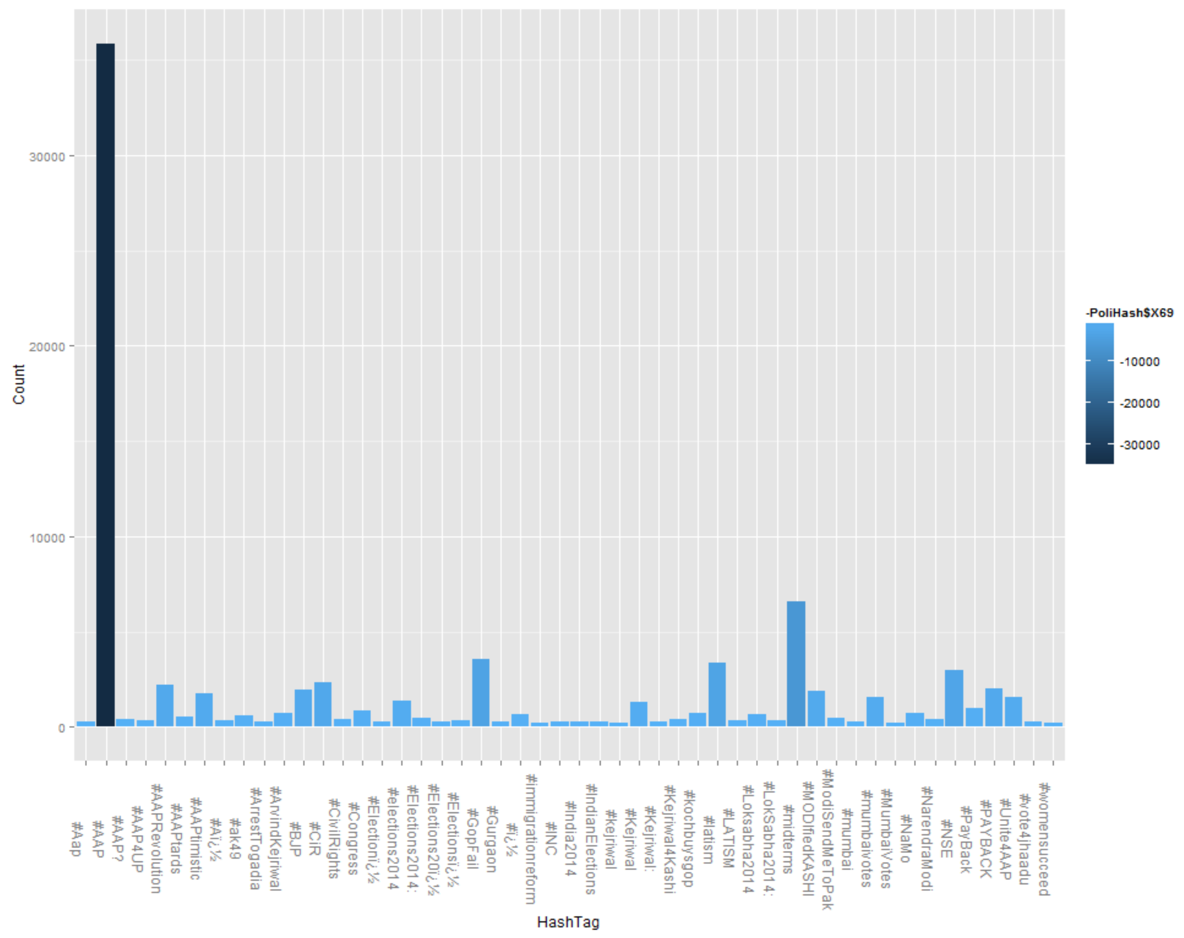
6.1.3 Hash tag counts

We get the most trending hash tags in this part.




Count
20000
15000
10000
5000
0
HashTag
#ApneMunde
#Binny
#Calendar
#contest
#cricket
#Cricket
#csk
#Csk
#CSK
#CSKvDD
#CSKvsDD
#DD
#DDvsCSK
#Dhoni
#Duminy
#giveaway
#ipl
#IPL
#IPL2014
#ipl7
#IPL7
#Jadeja
#Kallis
#KP
#KXIP
#livecricket
#MI
#Miller
#MomMotto
#news
#News
#Pepsi
#pepsiipl
#PepsiIPL
#playerbattles
#RiSE
#RR
#RRvsKXIP
#sports
#Sports
#SPORTS
#SRH
#Steyn
#sweeps
#Sweeps
#Warner
#whistlepodu
#WhistlePodu
#win
#Yuvraj
-apr21Hash$X183
-5000
-10000
-15000


Count
40000
30000
20000
10000
0
HashTag
#athletes
#BCCI
#cricket
#Cricket
#CSK
#DD
#Dhoni
#DilSeMI
#DishTwicket
#eit20
#giveaway
#ipl
#IPL
#IPL2014
#ipl7
#IPL7
#iplt20
#Jadeja
#KKR
#KP
#kxip
#KXIP
#KXIPvRR
#KXIPvsRR
#maxwell
#Maxwell
#MI
#Miller
#MomMotto
#news
#Pepsi
#pepsiipl
#pepsiIPL
#PepsiIPL
#PepsiIPL2014
#playerbattles
#RCB
#RR
#RRvKXIP
#RRvsKXIP
#sports
#Sports
#Steyn
#sweeps
#Sweeps
#Warner
#Watson
#WeAreTheRoyals
#win
#Yuvraj
-apr20Hash$X513
-10000
-20000
-30000
-40000

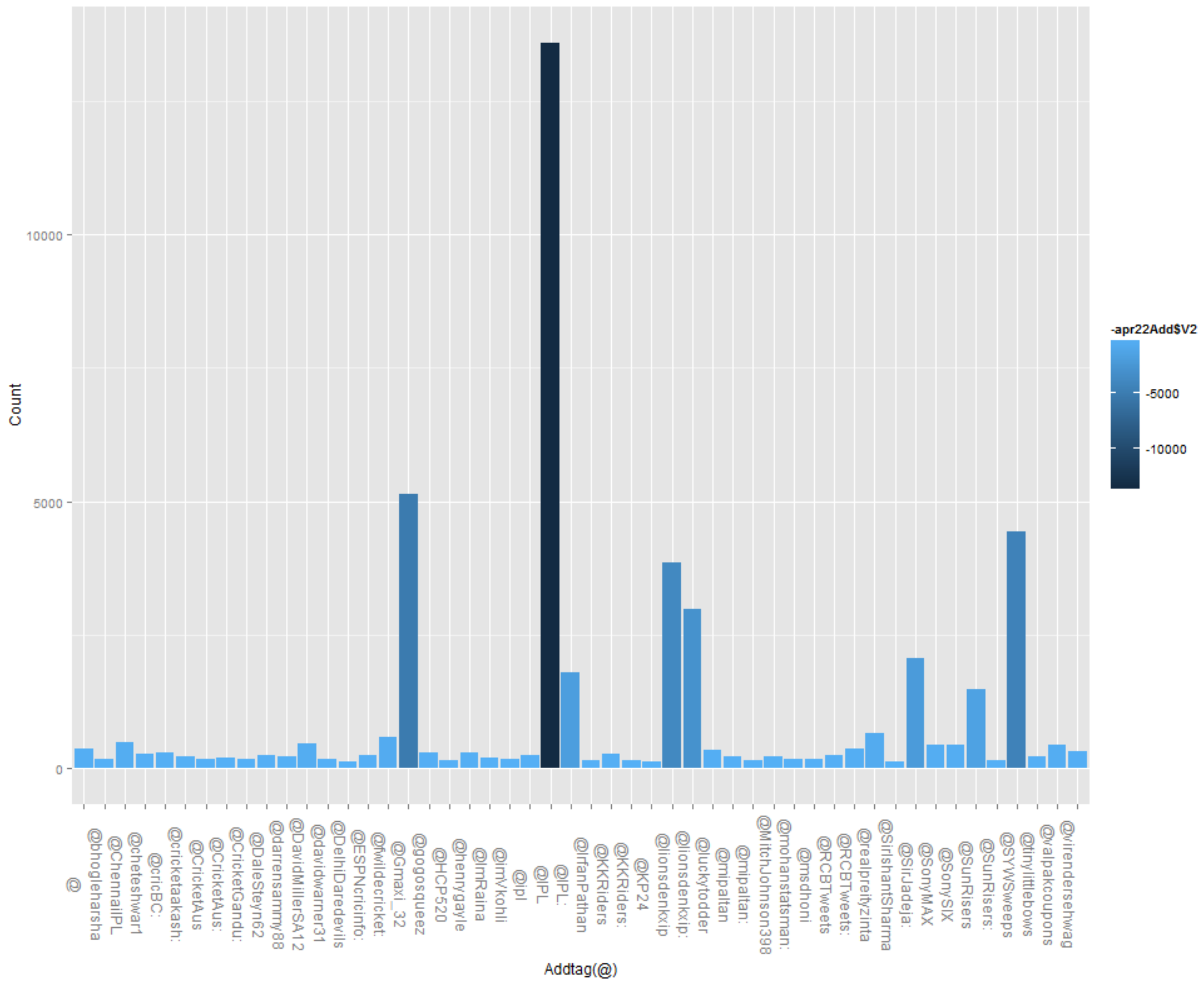


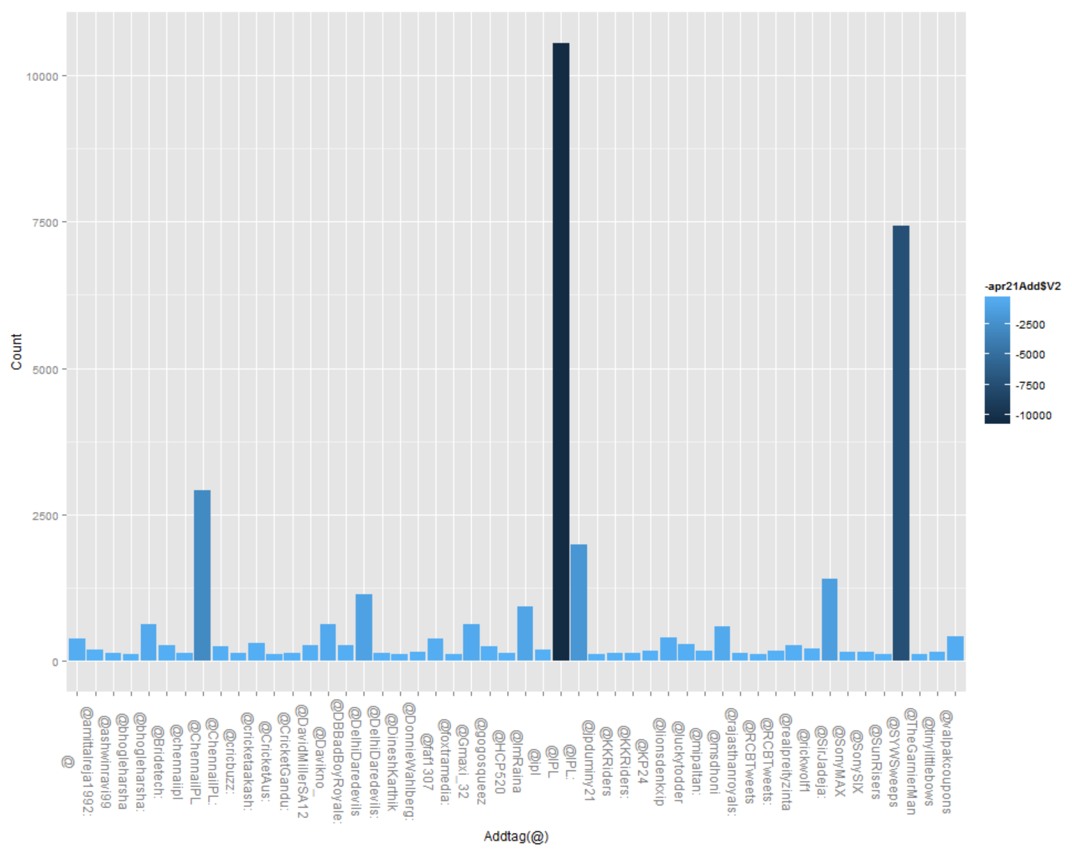
We can again see a drastic change of tweets Post IPL and Pre IPL times. Post IPL, it was all CSK, DD and other teams.

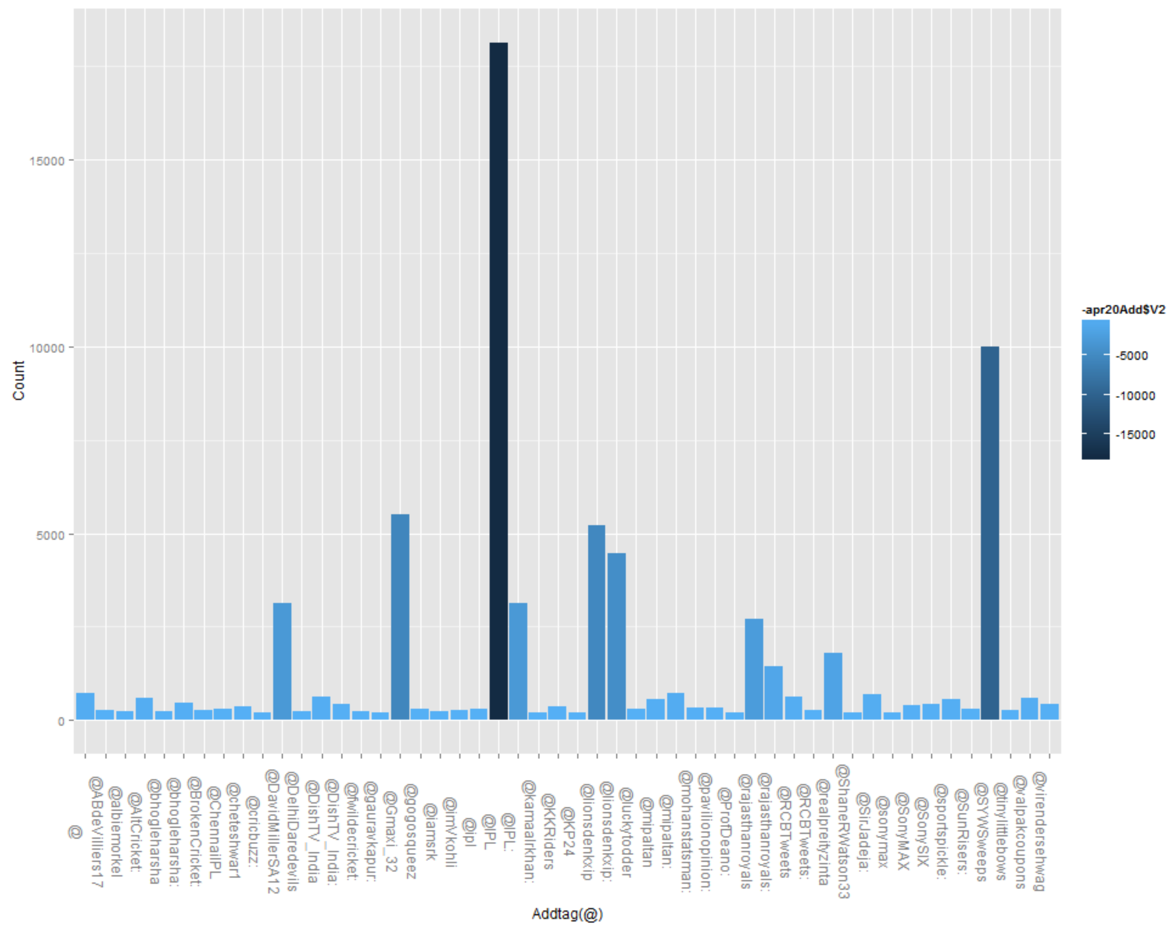
Pre IPL we can see the tweets were majorly occupied by BJP, Congress, Rahul Gandhi and Narendra Modi.

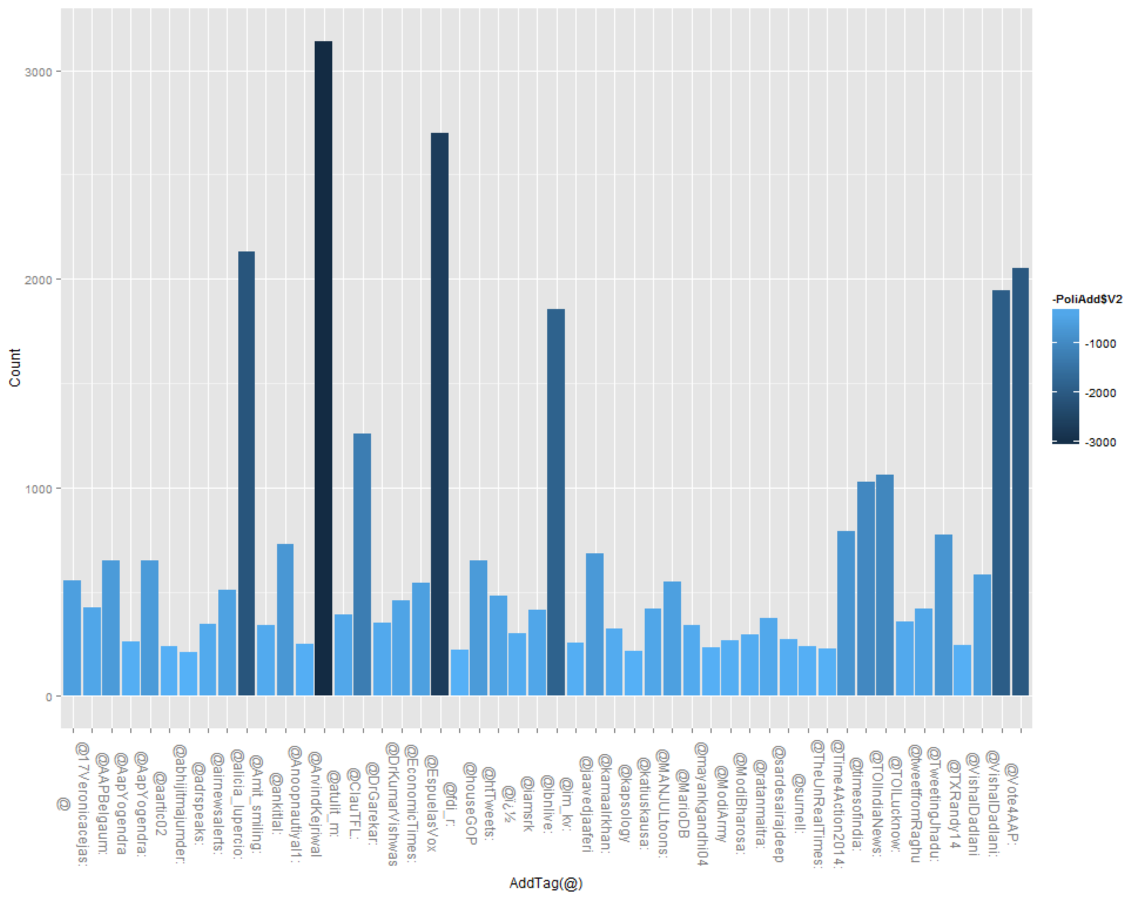
6.1.4 @User Counts

We get the most tweeted people in this section.




Count
10000
7500
5000
2500
0
Addtag(@)
@
@amittalreja1992:
@ashwinravi99
@bhogleharsha
@bhogleharsha:
@Bridetech:
@chennaiipl
@ChennaiIPL
@ChennaiIPL:
@cricbuzz:
@cricketaakash:
@CricketAus:
@CricketGandu:
@DavidMillerSA12
@Davikno_
@DBBadBoyRoyale:
@DelhiDaredevils
@DelhiDaredevils:
@DineshKarthik
@DonnieWahlberg:
@faf1307
@foxtramedia:
@Gmaxi_32
@gogosqueez
@HCP520
@ImRaina
@ipl
@IPL
@IPL:
@jpduminy21
@KKRiders
@KKRiders:
@KP24
@lionsdenkxip
@luckytodder
@mipaltan:
@msdhoni
@rajasthanroyals:
@RCBTweets
@RCBTweets:
@realpreityzinta
@rickwolf1
@SirJadeja:
@SonyMAX
@SonySIX
@SunRisers
@SYWSweeps
@TheGarnierMan
@tinylittlebows
@valpakcoupons
-apr21Add$V2
-2500
-5000
-7500
-10000


Count
15000
10000
5000
0
Addtag(@)
@
@ABdeVilliers17
@albiemorkel
@AltCricket:
@bhogleharsha
@bhogleharsha:
@BrokenCricket:
@ChennaiIPL
@cheteshwar1
@cricbuzz:
@DavidMillerSA12
@DelhiDaredevils
@DishTV_India
@DishTV_India:
@fwildecricket:
@gauravkapur:
@Gmaxi_32
@gogosqueez
@iamsrk
@imVkohli
@ipl
@IPL
@IPL:
@kamaalrkhan:
@KKRiders
@KP24
@lionsdenkxip
@lionsdenkxip:
@luckytodder
@mipaltan
@mipaltan:
@mohanstatsman:
@pavilionopinion:
@ProfDeano:
@rajasthanroyals
@rajasthanroyals:
@RCBTweets
@RCBTweets:
@realpreityzinta
@ShaneRWatson33
@SirJadeja:
@sonymax
@SonyMAX
@SonySIX
@sportspickle:
@SunRisers:
@SYWSweeps
@tinylittlebows
@valpakcoupons
@virendersehwag
-apr20Add$V2
-5000
-10000
-15000



Here we can see that if we divide it into two phases, then the only people who were tweeted were either Narendra Modi or Arvind Kejriwal in politics and Kohli, Yuvi, Dhoni, Maxwell in IPL cricket.

6.2 Implementing word-co-occurrence algorithm

We find the co-occurrence of hash tags in a single tweet. There are two ways to compute it, Pairs and Stripes. We also find the relative frequency for co-occurring hash tags.

6.2.1 Co-occurrence pair count

Algorithm –

Driver Class –

1. Initialize job.
2. Set attributes like mapper class, reducer class, output keys and values, number of reduced tasks and partitioner.
3. Run mapper
4. Run partitioner
5. Run reducer

Mapper Class –

1. Tokenize the line.
2. Check if the token starts with "#"
3. Store in an arraylist.
4. If size of arraylist is greater than 1.
5. Then for(i = 0 to size)
6. For(j=i+1 to size)
7. Emit(arralist(i) + arraylist(j), 1)

Partitioner class –

1. If the 1st character of hash tag lies between A-G,
    a. Then return part1
2. If the 1st character of hash tag lies between H-N,
    a. Then return part2
3. Else
    a. Return part0.

Reducer class –

1. Assign sum = 0
2. For all values of a token
    a. Sum = sum + value.
3. Emit(key,sum)

6.2.2 Co-occurrence stripe count
Algorithm –

Driver Class –

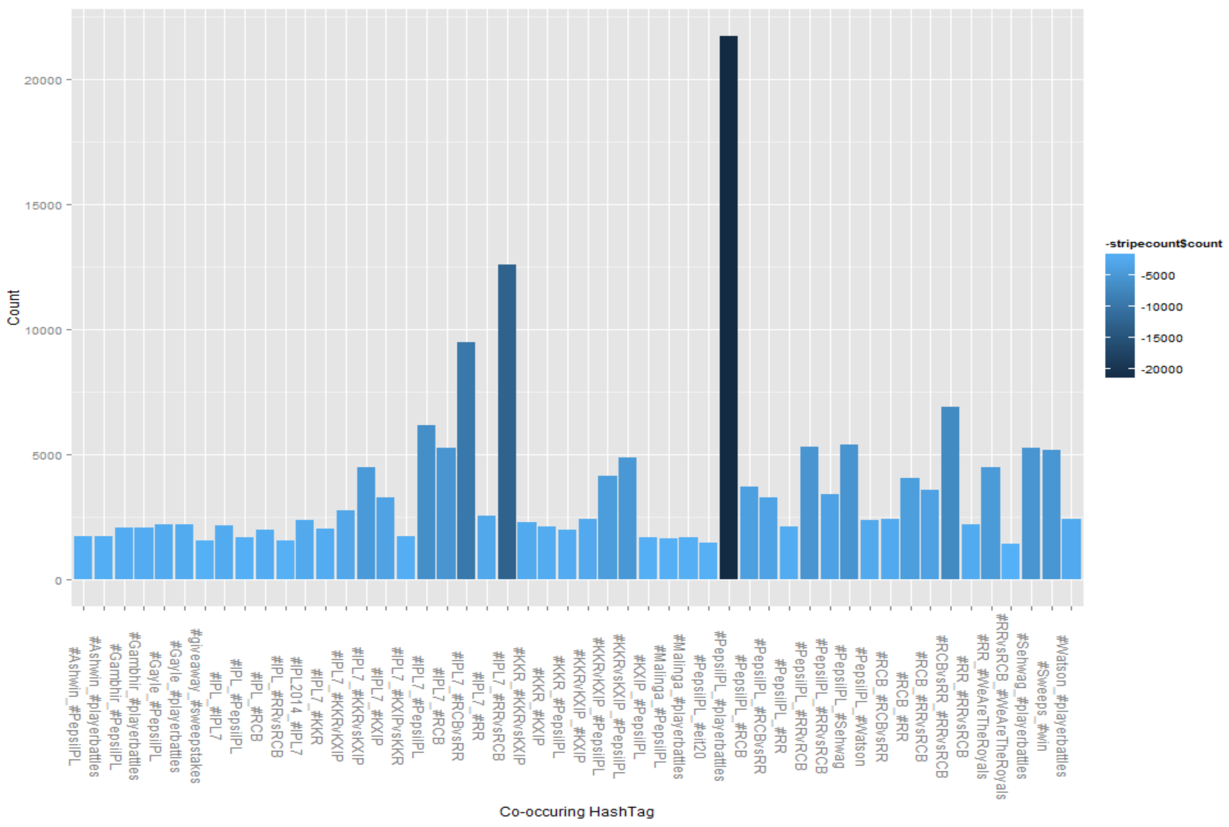
1. Initialize job.
2. Set attributes like mapper class, reducer class, output keys and values, number of reduced tasks and partitioner.
3. Run mapper
4. Run reducer

Mapper Class –

1. Tokenize the line.
2. Check if the token starts with "#"
3. Store in an arraylist.
4. If size of arraylist is greater than 1.
5. Then for(i = 0 to size)
6. For(j=i+1 to size)
7. Emit(arraylist(i), Map(arraylist(j),1))

Reducer class –

1. Aa<- new AssociativeArray
2. for all stripe S in stripes [S1 , S2 , S3 , . . .] do
   Sum(Aa , S)
3. for all terms x in stripe Aa
   Emit(term w_x, x.value)



Here we see that maximum co-occurring hash tags were IPL and Player battles. We wont see any politics and crickets hash tags occurring together. So we can say that only cricket related tweets were occurring together.

6.2.3 Co-occurrence pair RF

Algorithm –

Driver Class –

1. Initialize job.
2. Set attributes like mapper class, reducer class, output keys and values, number of reduced tasks and partitioner.
3. Run mapper
4. Run partitioner
5. Run reducer

Mapper Class –

1. Tokenize the line.
2. Check if the token starts with "#"
3. Store in an arraylist.
4. If size of arraylist is greater than 1.
5. Then for(i = 0 to size)
6. For(j=i+1 to size)
7. Emit(arraylist(i) , 1)
8. Emit(arraylist(i) + arraylist(j), 1)

Partitioner class –

1. If the 1st character of hash tag lies between A-G,
    b. Then return part1
2. If the 1st character of hash tag lies between H-N,
    c. Then return part2
3. Else
    a. Return part0.

Reducer class –

1. Assign sum = 0
2. For all values of a token
    a. Sum = sum + value.
3. If single hash, then add in Map
4. Else get total sum of 1st hash
5. Emit(key,sum/total sum)

6.2.4 Co-occurrence stripe count

Algorithm –

Driver Class –

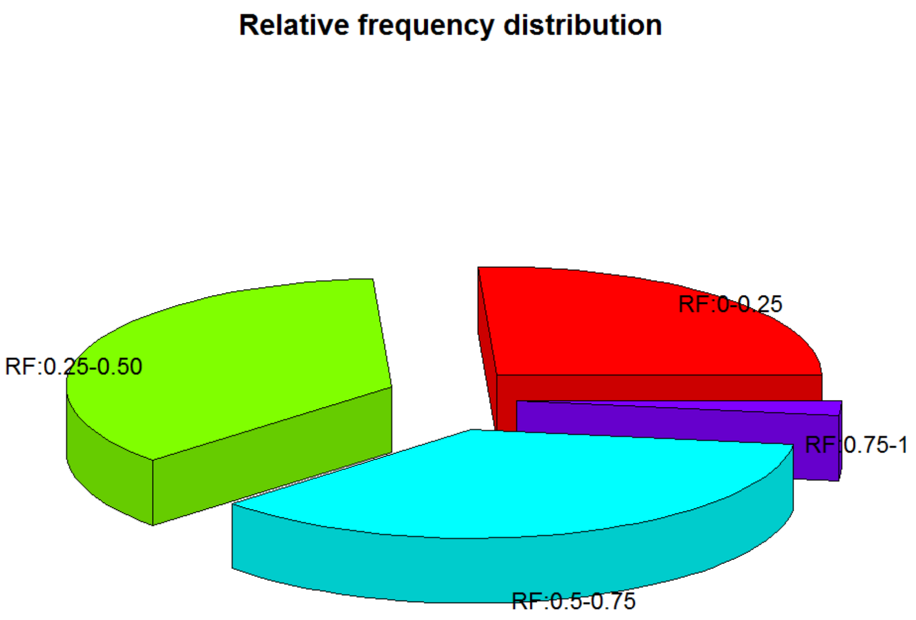
1. Initialize job.
2. Set attributes like mapper class, reducer class, output keys and values, number of reduced tasks and partitioner.
3. Run mapper
4. Run reducer

Mapper Class –

1. Tokenize the line.
2. Check if the token starts with "#"
3. Store in an arraylist.
4. If size of arraylist is greater than 1.
5. Then for(i = 0 to size)
6. For(j=i+1 to size)
7. Emit(arraylist(i), Map(arraylist(j),1))

Reducer class –

4. Aa<- new AssociativeArray
5. for all stripe S in stripes [S1 , S2 , S3 , . . .] do
   Sum(Aa , S)
6. for all terms x in stripe Aa
   Emit(term w_x, x.value/total sum of stripe Aa)



Here we can see that major chunk of co-occurring tweets fall in the relative frequency of 0.5-0.75. So we can say that the tweets are pretty evenly distributed.

6.3 Clustering: Map reduce to implement K-Means

Clustering on follower count with three clusters “LOW”, “MID” and “HIGH”. Starting with the initial cluster value (100, 1000, 100000), K-means Algorithm is executed on Hadoop till convergence. Cluster centroids are stored in three files corresponding to every cluster and during every iteration these values are appended with new cluster centroid. Counter is used to keep track of convergence and if convergence is not reached counter is increased and convergence will be achieved when counter value will be 0 for a particular job.

Mapper job: Mapper will get follower count and assign it to one of the three cluster “LOW”, “MED” and “HIGH” and will emit <clustered, follower count>

Reducer job: Reducer will calculate new centroids with the mean of follower count and also will check if the new value is equal to old value or not thus increasing the counter f applicable.

Algorithm:

Main class (Kmeans.class)

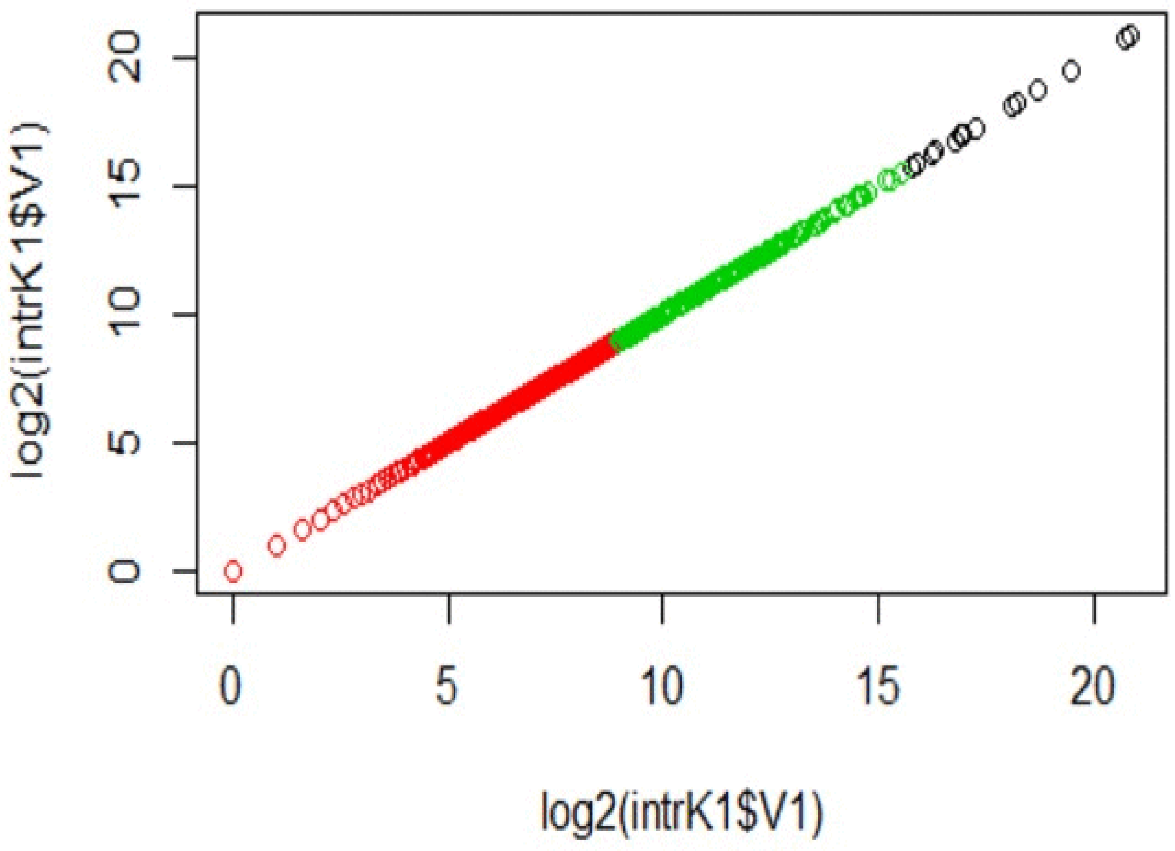
1. Declare counter enums
2. While(true)
3. Run Mapper
4. Run Reducer
5. Delete input file
6. Copy output file to input path
7. If(counter==0)
8. Break;
9. While end

Mapper class:

1. Split the input to get nodes, distance and adjucancy list.
2. If(parent node)
3. Emit<node,dist>
4. If(node of adjacency list>
5. Emit< adjacency node, distance+1>

Reducer class:

1. Calculated min distance for node key
2. Check for convergence and manipulate counter
3. Emit<node,min>



We can see that different colors show that different clusters.
Red – Low
Green – Medium
Black – High

This data has converged after 8 iterations.

6.4 Shortest path Algorithm using Map Reduce

Minimum path is calculated on edges of a graph and Algorithm is executed till convergence is achieved.
Mapper main function: emit dist of node and its adjacency list.
Reduced main function: calculate min distance from all the distance pertaining a node and check for convergence by manipulating counter.
Counter is used to check for convergence.

Algorithm:

Main class (Dijikstra.class):

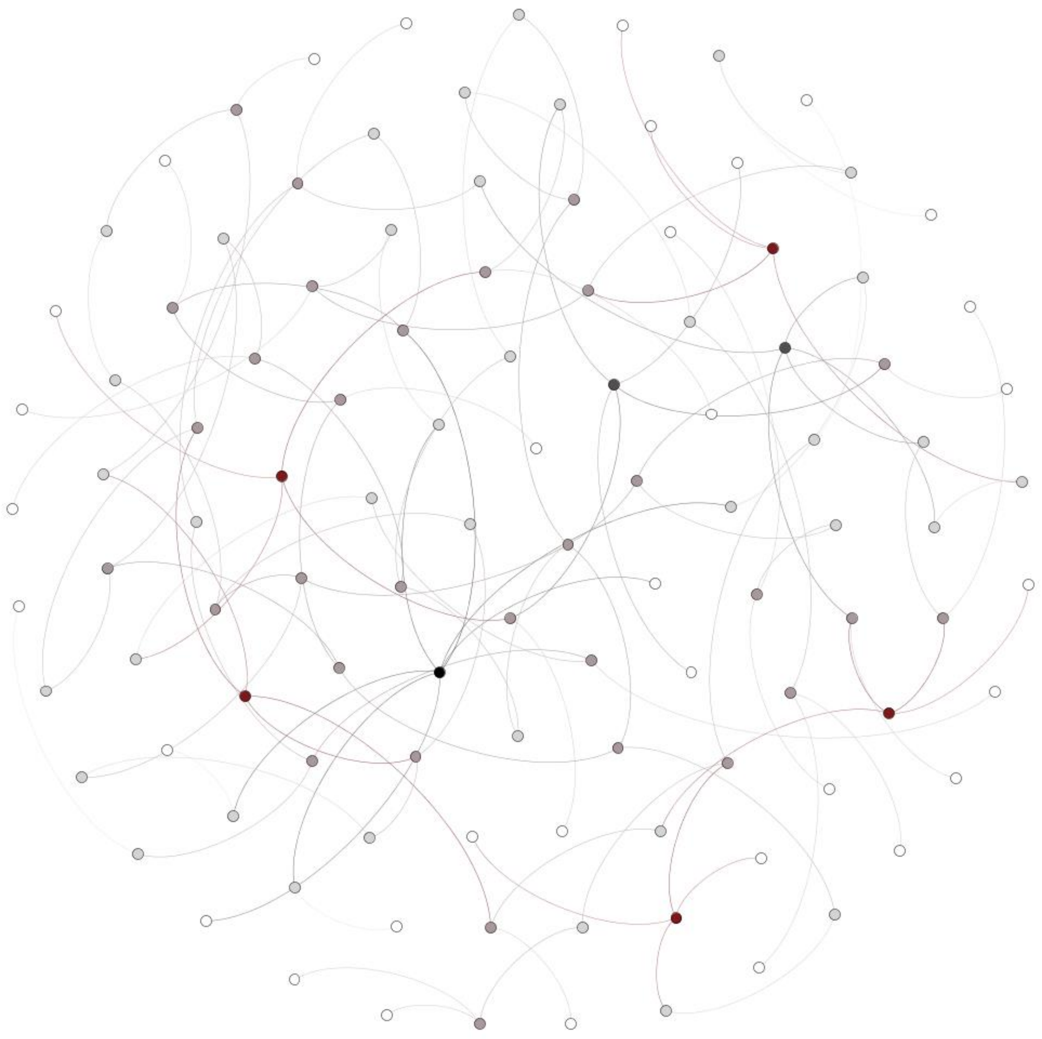
1. Declare counter enums
2. While(true)
3. Run Mapper
4. Run Reducer
5. Delete input file
6. Copy output file to input path
7. If(counter==0)
8. Break;
9. While end

Mapper class:

1. Read cluster centres
2. Assign incoming follower count to one cluster by calculating abs distance
3. Emit< clustered, follower count>

Reducer class:

1. Calculate mean (new cluster centre by calculating mean of all follower count)
2. Read old cluster values from files
3. Increment counter if convergence is not achieved
4. Emit<follower count, clusterID)



This is the graph generated using GEPHI for Dijikstra's algorithm. We can see the edges and the connection between the edges.
The colored edges show that there are more connections to that edge.

7. <u>Lessons learnt</u>:

- We learnt basics of Map Reduce and working of it.
- We got hands on experience on handling large data, do analytics on it and filter out meaningful data out of loads of data.
- We learnt how to configure and install HDFS and Hadoop 2.0.
- We learnt working on tools like GEPHI to create graphs.
- We learnt how powerful map reduce is, and that it can do wonders with right data provided.
- We also built upon our learning on R and used to it to a good effect here, to show proper analysis of data.