OPIM 5503 - DATA ANALYTICS USING R

DATA KNIGHTS

Jaya Chandra Balusu Yeswanth Yenisetti Raja Vemuri
Leela Kishore Donthireddy Shikha Kanwar

Social Media Analytics

In this project, we developed a system to analyze sentiments and public opinions about US Presidential Candidates. The system utilizes semantic features of English language to assign polarities to words and give a sense of public sentiments. We have used twitter data for this analysis. Over years, Twitter has evolved from yet another social networking site to a very powerful microblogging platform which by its nature entails highly specific opinions in few words. We have utilized this open mine of information to build data for our analysis. Sentiment Analysis, is the process of investigating opinion of speakers and classifying emotions into positive, negative and neutral buckets. Topic modeling is used to understand the current trending events. We have used this concept to study various sentiments related to upcoming US Presidential elections and associated candidate. The insights from these studies can be very illuminating for the stakeholders as well as for analysts, who are trying to decipher the mystery beforehand. The process to build the system relies on core concepts of text mining also known as opinion mining which aims to extract numeric information from unstructured text and make it suitable for various text/data mining algorithms.

To illustrate the methodology and conclusions derived from this study, this paper has been divided into following broad categories:

A. Access tweets using twitter API

B. Preprocess raw text data and make it suitable for data mining/mathematical analysis

C. Assign polarities to words in each tweet and calculate polarity scores

D. Topic Modeling

E. Present results visually and derive conclusions

**Before executing the code please set your working directory Eg: setwd("C:/Spring Semester/Data Analytics using R/Project/") Place the given Negative and Positive words files in the Working directory

A: Access tweets using twitter API

Install and load required packages for preprocessing and analyzing tweets

twitteR package includes set of text analytics functions. In this project, this library is primarily used to provide interface to Twitter web API.

```
library(twitteR)
```

tm package is a framework of text mining applications, it provides variety of functions to process raw tweets into high quality information from text.

```
library(tm)
```

```
## Loading required package: NLP
```

httr package is a collection of tools for working with URLs and HTTP while accessing online data.

```
library(httr)
```

```
##
## Attaching package: 'httr'
```

```
## The following object is masked from 'package:NLP':
##
##     content
```

This package is used to create word clouds.

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

The stringr package provides functions for string processing, the additional functionalities that which are missing in R base package

```
library(stringr)
```

PLYR package is used for split-apply-combine (SAC) procedures i.e. applying functions over grouped data.

```
library(plyr)
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:twitteR':
##
##     id
```

topicmodels package is used to perform topic modelling using LDA and CTM algorithms

```
library(topicmodels)
```

Set up connection to Twitter search API for downloading tweets.

Following is the step by step process: ->sign up for twitter and create an application in https://apps.twitter.com/ (https://apps.twitter.com/) -> save the consumer/API keys and consumer/API secret key and access tokens

Enter the consumer key provided by Twitter application in api_key

Enter the consumer secret key supplied by Twitter in api_secret
Enter the access token in access_token

Enter access token secret supplied by Twitter in access_token_secret

Set up authentication using following function that uses consumer keys and access tokens “setup_twitter_oauth function” is part of {twitteR} library and uses OAuth authentication handshake function from httr package for a twitter session

```
setup_twitter_oauth(api_key,api_secret,access_token,access_token_secret)
```

```
## [1] "Using direct authentication"
```

B: Preprocess raw text data and make it suitable for data mining/mathematical analysis

Extract Tweets and create a dataframe for processing

The Twitter search API allows access to a subset of recent tweets for querying. For this project, we are extracting tweets related to four candidates: two Republicans and two Democrats competing for US Presidential elections.

To avoid re-running basic preprocessing steps four times a function ('tweet') is designed to return a clean corpus from raw tweets. The two arguments that user need to provide for this function are: seachstring - this is a search query to twitter. Twitter searches for all tweets related to this string nt- takes number of tweets required

Function SearchTwitter searches for all tweets matching with search string provided

```
tweet=function(searchstring,nt)
  {
  tweets = searchTwitter(searchstring, n=nt,lang="en")

#This function takes a list of tweets returns a data.frame version of the tweet
s

    tweets_dataframe = twListToDF(tweets)

# The data at this point has 16 columns: text, favorited, favoriteCount, replyT
oSN,created, truncated, replyToSID, ID, replutoUID...etc.for the sentiment anal
ysis exercise we will  retain and work on text captured in this dataframe
#
#4 Data preprocessing and Text mining
# Text mining involves the process of deriving meaningful information from raw
text data
# Twitter data is highly unstructured form of text data, primarily because it i
s an informal
# communication which includes presence of undesirable content like expression
s,slang's and high frequency of stop words. The typos and messaging short-form
s make it more difficult to analyze. The data cleaning process aims at removin
g all the inconsistencies and unwanted words in text that are not useful for an
alysis.

# gsub function comes in handy for performing various data cleaning tasks
# gsub searches for matches to argument pattern within each element of a charac
ter vector and replaces all occurrences of first argument with the second argum
ent.
# gsub can also  take regular expressions like [[:digit:]]
# We will use GSUB for follow operations:

#a. remove special characters and retain alpha numeric, '//','(quote)and '@' fr
om tweets text

    tweets_data = gsub("[^0-9A-Za-z@///' ]", "", tweets_dataframe$text)



#b. while some of the text is from re-tweets, we would certainly like to get ri
d of name of the person whose post was re-tweeted. This will remove word 'RT' a
nd the name of person
#w=mentioned after RT

tweets_data = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "",tweets_data)

#c. The other form of noise in twitter data is names of people between the tex
t.
#The function below removes name of people by matching anything that starts wit
h @ symbol as name
```

```
    tweets_data = gsub("@\\w+", "", tweets_data)

#d.  Since the purpose of sentiment analysis is to analyze words and not number
s we can remove #the numeric digits between the text using function gsub

    tweets_data = gsub("[[:digit:]]", "", tweets_data)

# Data obtained from internet usually contains a lot of html entities and link
s;which gets
# embedded in the raw  data. It is important to get rid of these entities as a
part of data #sanitization process. The function below matches http links and r
emoves them
tweets_data = gsub('http\\S+\\s*',"", tweets_data)

# Create a corpus and prepare data for analysis

# corpus is a collection of large structured texts and various data preprocessi
ng functions
# available in TM package can be applied on corpus only.
# Following function converts dataframe to corpus

    tweets_corpus  <-  Corpus(VectorSource(tweets_data))


# To create a consistent pattern we will convert all words in text to lowercas
e. for example, #frequency functions otherwise would count same word in differe
nt case as two different words

tweets_corpus<-  tm_map(tweets_corpus,  tolower)

#create a plain text document using content in corpus

tweets_corpus  <-  tm_map(tweets_corpus,  PlainTextDocument)


#All the punctuation marks between text should be removed

tweets_corpus  <-  tm_map(tweets_corpus,  removePunctuation)


#While carrying out text analysis driven at the word level, it is required to r
emove the commonly #occurring words also called stop-words. One approach is to
create a create a long list of #stop-words and other is too use a predefined la
nguage specific libraries
#We will apply both approaches:
#1) use common stop words for english language

    tweets_corpus  <-  tm_map(tweets_corpus,  removeWords,  stopwords("english"))
```

```
#2) based on our observation of data, we will create a customized list to be ap
plied on top of #the text cleaned by using predefined stop words library

    st  =  c("amp","you'll","you'd","tedcruz","she","yet","hilari","trump","cru
z",
           "you", "yourself", "won't", "wont","will","hillari","hilaryclinton",
           "donald","bernie","berniesanders","sen","ted","cruz","hillary",
           "clinton","i", "me", "my", "myself", "we", "our", "ours", "ourselve
s",
           "you", "your", "yours", "yourself", "yourselves", "he", "him", "hi
s",
           "himself", "she", "her", "hers", "herself", "it", "its", "itself",
           "they", "them", "their", "theirs", "themselves", "what", "which",
           "who", "whom", "this", "that", "these","fuck","those", "am", "is",
"are",
g",        "was", "were", "be", "been", "being", "have", "has", "had", "havin

e",
e'd",      "do", "does", "did", "doing", "would", "should", "could", "ought",
           "i'm", "you're", "he's", "she's", "it's", "we're", "they're", "i'v
n't",
           "you've", "we've", "they've", "i'd", "you'd", "he'd", "she'd", "w
n't",
           "they'd", "i'll", "you'll", "he'll", "she'll", "we'll", "they'll",
o's",      "isn't", "aren't", "wasn't", "weren't", "hasn't", "haven't", "had

w's",      "doesn't", "don't", "didn't", "won't", "wouldn't", "shan't", "should
l",
en",
           "can't", "cannot", "couldn't", "mustn't", "let's", "that's", "wh

o",
           "what's", "here's", "there's", "when's", "where's", "why's", "ho

ain",
           "a", "an", "the", "and", "but", "if", "or", "because", "as", "unti


           "while", "of", "at", "by", "for", "with", "about", "against", "betwe
r",
           "into", "through", "during", "before", "after", "above", "below", "t
an",
           "from", "up", "down", "in", "out", "on", "off", "over", "under", "ag


           "further", "then", "once", "here", "there", "when", "where", "why",
           "how", "all", "any", "both", "each", "few", "more", "most", "othe


           "some", "such", "no", "nor", "not", "only", "own", "same", "so", "th
           "too", "very")
    tweets_corpus <- tm_map(tweets_corpus, removeWords, st)
```

```
# Word stemming means to reduce word to its root form i.e. to remove any tense
information or to #convert derived words to their root form. TM library functio
ns allows us to stem the words and #thus identify words with same meaning whic
h  might otherwise look different to machine
    tweets_corpus  <-  tm_map(tweets_corpus,stemDocument)

    return(tweets_corpus)

}
```

tweets function can be dynamically used to retrieve and process tweets. This avoids writing same processing step four times for each search string. Extract data using tweet function and save in a clean corpus for four candidates: Hilary Clinton, Donald Trump, Ted Cruz and Bernie Sanders

```
bs=tweet("berniesanders",200)
dt=tweet("realDonaldTrump",200)
hc=tweet("Hilary",200)
tc=tweet("tedcruz",200)
```

Retrieve and visualize the most frequent words associates/used for these 4 candidates.

These results can be very illuminating as they can help candidates understand the public notion. In terms of preparing data for this analysis, we will convert corpus into a matrix that will identify the most frequent word from each text line and number of times the word occurs.

tm library provides ‘DocumentTermMatrix’ function to achieve this data transformation

DocumentTermMatrix which is matrix that describes frequency of words that occur in collection of documents (or text in this case). Each row is a document (or text ID) in corpus and each column corresponds to word/term. The value of column is 1 if word exists in document otherwise 0.

```
DTM_hc  =  DocumentTermMatrix(hc)
DTM_dt  =  DocumentTermMatrix(dt)
DTM_bs  =  DocumentTermMatrix(bs)
DTM_tc  =  DocumentTermMatrix(tc)
```

The sum of columns will give number of times each word has been repeated in all the tweets collected for each candidate.

```
freq_hc <- colSums(as.matrix(DTM_hc))
freq_dt <- colSums(as.matrix(DTM_dt))
freq_bs <- colSums(as.matrix(DTM_bs))
freq_tc <- colSums(as.matrix(DTM_tc))
```

Create a data frame with two columns word and frequency

```
DTM_hc <- data.frame(word=names(freq_hc), freq=freq_hc)
DTM_dt <- data.frame(word=names(freq_dt), freq=freq_dt)
DTM_bs <- data.frame(word=names(freq_bs), freq=freq_bs)
DTM_tc <- data.frame(word=names(freq_tc), freq=freq_tc)
```

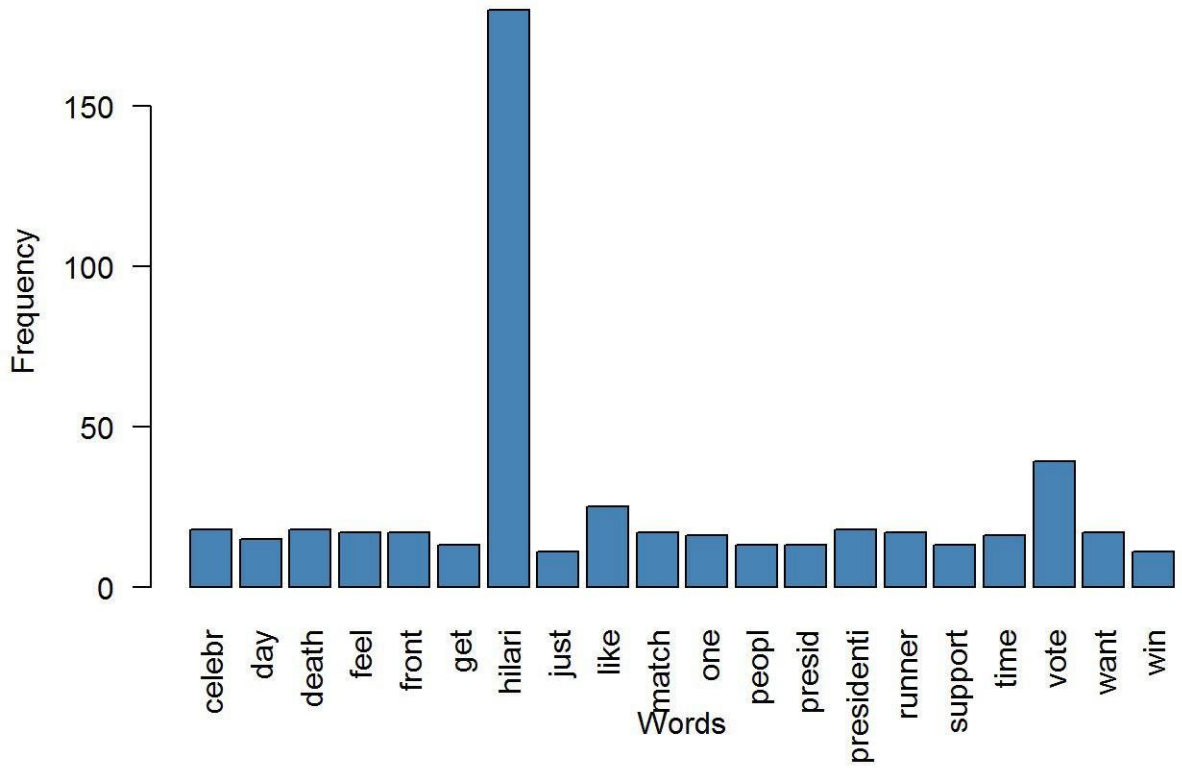
Bar plots and word clouds convey the message associated with numbers derived above. They are produced by comparing the most frequent words associated with each candidate

Minimum frequency can be entered here:

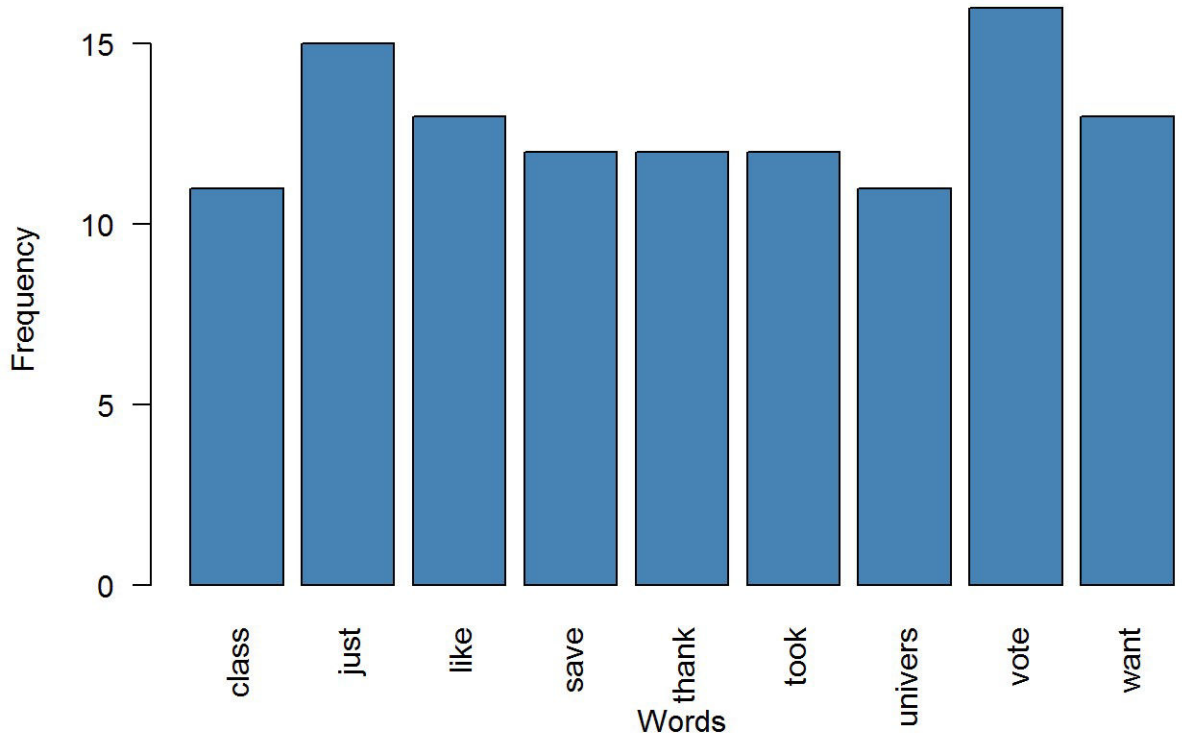
```
min freq =10
```

For example you want to plot only words which occur more than 100 times assign 100 to min_freq

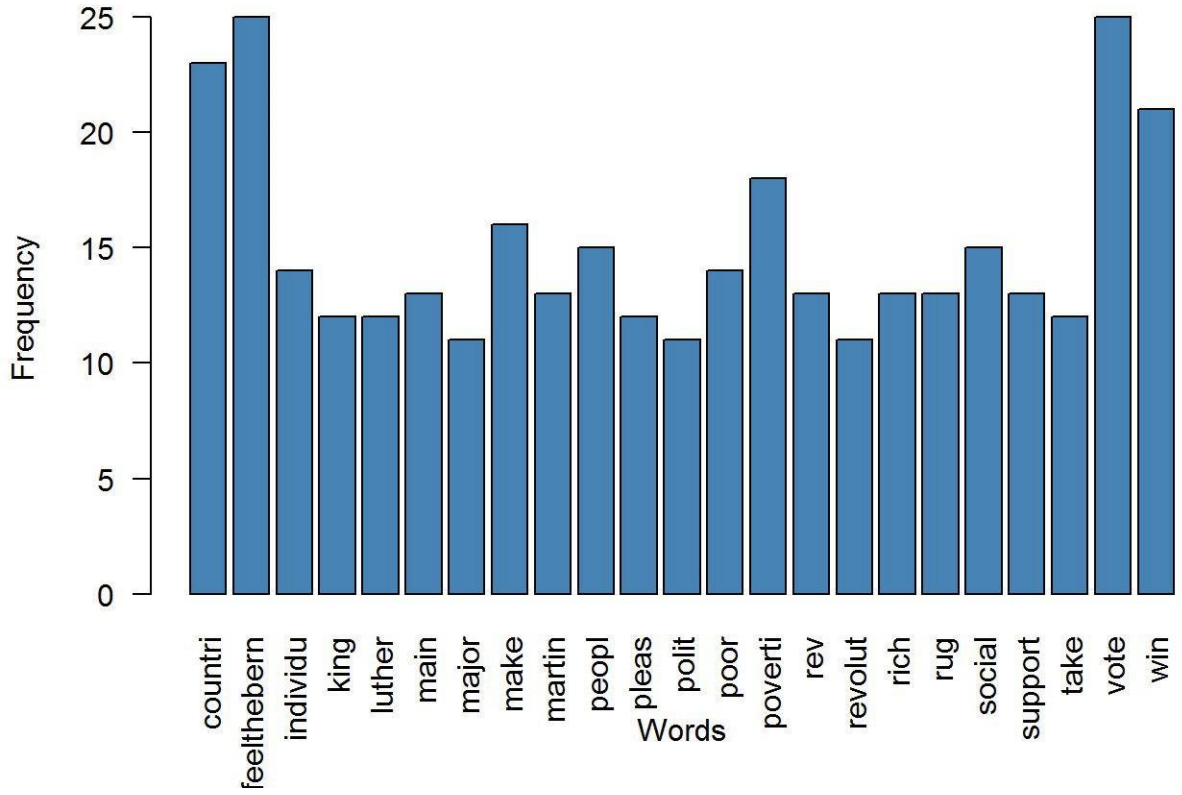
```
barplot(DTM_hc$freq[DTM_hc$freq>min_freq],names.arg=DTM_hc$word[DTM_hc$freq>min
_freq],xlab="Words", ylab="Frequency",col="steelblue",las=2)
```



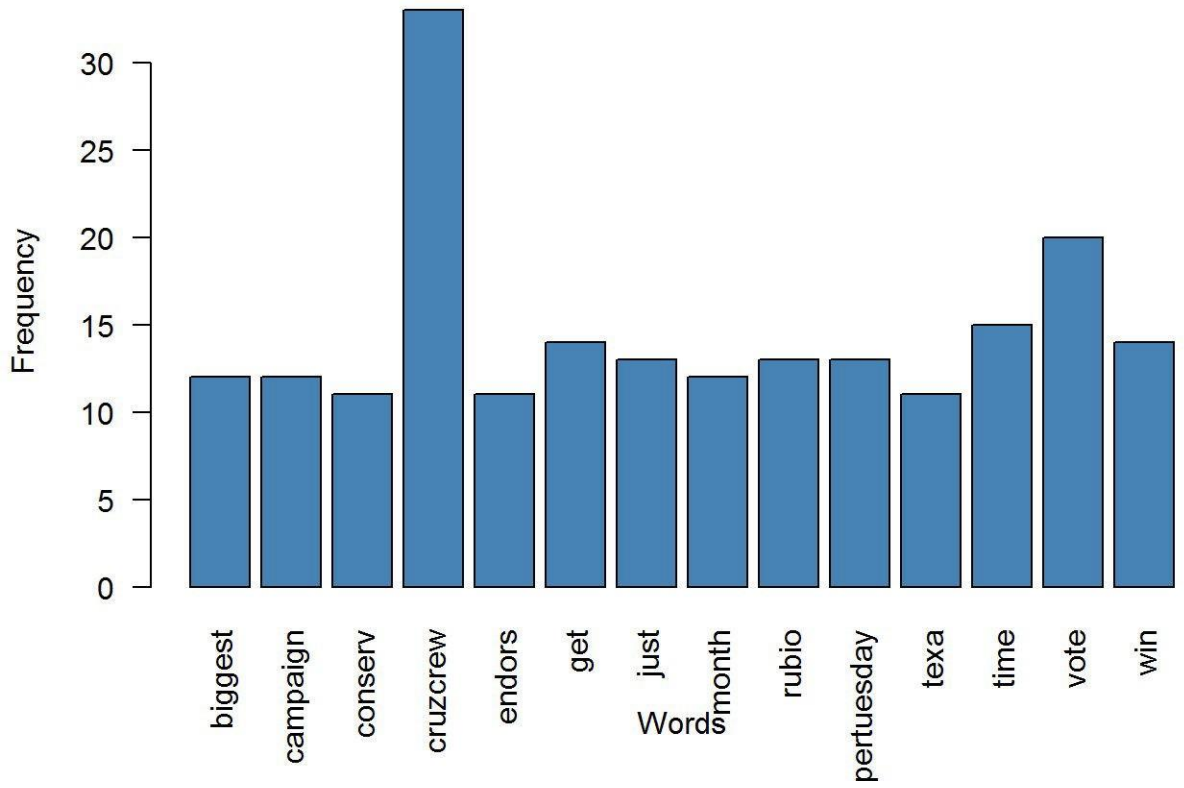

```
barplot(DTM_dt$freq[DTM_dt$freq>min_freq],names.arg=DTM_dt$word[DTM_dt$freq>min
_freq],xlab="Words",  ylab="Frequency",col="steelblue",las=2)
```




```
barplot(DTM_bs$freq[DTM_bs$freq>min_freq],names.arg=DTM_bs$word[DTM_bs$freq>min
_freq],xlab="Words",  ylab="Frequency",col="steelblue",las=2)
```




```
barplot(DTM_tc$freq[DTM_tc$freq>min_freq],names.arg=DTM_tc$word[DTM_tc$freq>min
_freq],xlab="Words",  ylab="Frequency",col="steelblue",las=2)
```

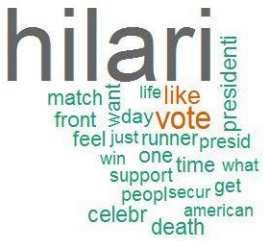


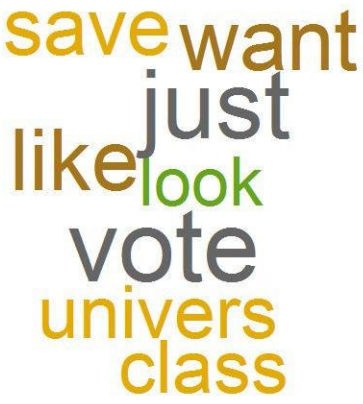
Use par option to compare four word clouds at a time

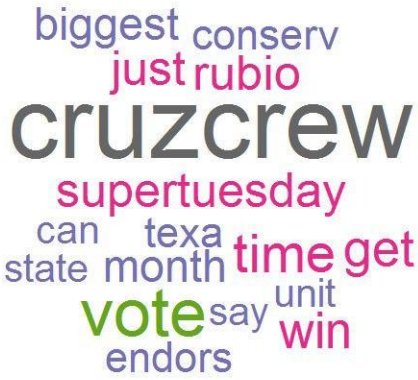
```
opar = par()
par(mfrow=c(2,2))

wordcloud(names(freq_hc),freq_hc,  min.freq=min_freq,colors=brewer.pal(8,  "Dark
2"))
wordcloud(names(freq_dt),freq_dt,  min.freq=min_freq,colors=brewer.pal(8,  "Dark
2"))
```

```
wordcloud(names(freq_bs),freq_bs,  min.freq=min_freq,colors=brewer.pal(8,  "Dark
2"))
```

```
wordcloud(names(freq_tc),freq_tc,  min.freq=min_freq,colors=brewer.pal(8,  "Dark
2"))
```








```
# These visualizations will help  perceiving, associations people make with eac
h candidate
```

```
par(opar)
```

```
## Warning in par(opar): graphical parameter "cin" cannot be set
```

```
## Warning in par(opar): graphical parameter "cra" cannot be set
```

```
## Warning in par(opar): graphical parameter "csi" cannot be set
```

```
## Warning in par(opar): graphical parameter "cxy" cannot be set
```

```
## Warning in par(opar): graphical parameter "din" cannot be set
```

```
## Warning in par(opar): graphical parameter "page" cannot be set
```

C: Assign polarities to words in each tweet and calculate polarity scores

Polarity Score

People use twitter as a microblogging platform to convey their negative as well as positive messages. Machines are not able to classify these words into literal negative and positive categories; so we have to use mathematical ways of assigning polarity to words.

For this exercise we have used ‘Bag of Words’ approach to identify polarity associated. This approach compares text against a large collection of negative and positive words databases and assign polarities to them.

Following function is designed to calculate polarity score. Polarities are denoted by +1 for positive words and -1 for negative word in the text. Polarity score is calculated for each tweet as sum of polarities.

For example if a tweet has 2 positive words and negative words and 5 neutral. The polarities assigned will be as (1, 1,-1,-1,-1, 0, 0, 0) and the Polarity score will be sum of polarities = -1

```
polarity_score = function(tweets, positive, negative)
{
    score =laply(tweets, function(tweet, positive, negative)
    {
        #before matching words in each tweet we will need to separate multiple
attached
        #words into individual entities. For example, "brokelaptop" to "broke l
aptop"
        #str_split function provides this functionality

        words = str_split(tweet,"\\s+")
        # unlist function simplifies a list structure i.e. is a sentence into v
ector which contains
        #all atomic components present in tweets
        words = unlist(words)

        # function matches the words with positive and negative databases and c
alculates score
        positive_overlap = match(words, positive)
        negative_overlap= match(words, negative)
        positive_overlap =!is.na(positive_overlap)
        negative_overlap= !is.na(negative_overlap)
        score =sum(positive_overlap) - sum(negative_overlap)
        return(score)
    }, positive, negative)
    scores =data.frame(score=score, text=tweets)
    return(scores)
}
```

Access the negative and positive library which will be used in the polarity score function above. Please change the path as per your location of files

```
positive <- scan('positive-words.txt', what='character', comment.char=';') #fil
e with positive words
negative <- scan('negative-words.txt', what='character', comment.char=';') #fil
e with negative words
```

Before applying function convert corpus to dataframe

```
hc_df=data.frame(text = sapply(hc, as.character), stringsAsFactors = FALSE)
dt_df=data.frame(text = sapply(dt, as.character), stringsAsFactors = FALSE)
bs_df=data.frame(text = sapply(bs, as.character), stringsAsFactors = FALSE)
tc_df=data.frame(text = sapply(tc, as.character), stringsAsFactors = FALSE)
```

Using the function defined above calculate Polarity score for each tweet for all candidates

```
Sentiment_scores_hc = polarity_score(hc_df$text, positive, negative)
Sentiment_scores_dt = polarity_score(dt_df$text, positive, negative)
Sentiment_scores_bs = polarity_score(bs_df$text, positive, negative)
Sentiment_scores_tc = polarity_score(tc_df$text, positive, negative)
```

Score less than zero means a negative polarity and a score greater than zero means positive, neutral if zero. Create polarity column with negative or positive or neutral as values

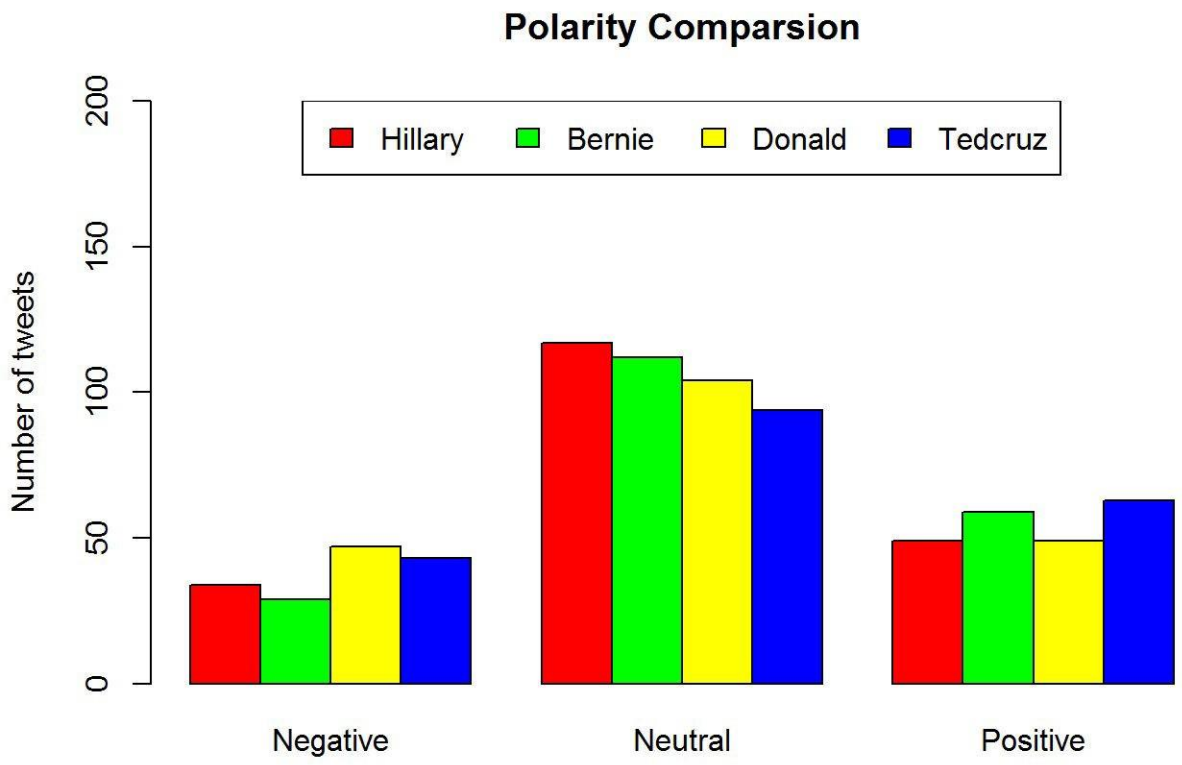
```
Sentiment_scores_hc$polarity=ifelse(Sentiment_scores_hc$score>0,"positive",   ife
lse(Sentiment_scores_hc$score<0  , "negative", "neutral"))
Sentiment_scores_dt$polarity=ifelse(Sentiment_scores_dt$score>0,"positive",   ife
lse(Sentiment_scores_dt$score<0  , "negative", "neutral"))
Sentiment_scores_bs$polarity=ifelse(Sentiment_scores_bs$score>0,"positive",   ife
lse(Sentiment_scores_bs$score<0  , "negative", "neutral"))
Sentiment_scores_tc$polarity=ifelse(Sentiment_scores_tc$score>0,"positive",   ife
lse(Sentiment_scores_tc$score<0  , "negative", "neutral"))
```

Prepare a table with frequency of negative, neutral and positive tweets. We've used table function to give the summary data and function t to transpose

```
x=   data.frame(table(Sentiment_scores_hc$polarity),table(Sentiment_scores_bs$pol
arity),table(Sentiment_scores_dt$polarity),table(Sentiment_scores_tc$polarity))
x = x[,c(2,4,6,8)]
colnames(x)<-c(  "Hillary","Bernie","Donald","Tedcruz")
x = t(x)
colnames(x)<-c("Negative", "Neutral",  "Positive")
```

Bar plot to compare polarities of Hillary, Donald, Bernie, TedCruz we have limited the y axis basing on number tweets downloaded so that the legend won't mix with the bars

```
colors = c("red",  "green","yellow","blue")
barplot(x,beside=T, col=colors,ylim=c(0,nrow(hc_df)),ylab="Number of tweets",ma
in="Polarity Comparsion")
legend("top",legend=rownames(x),horiz = TRUE,fill=colors)
```



To compare extreme sentiments it will be good to look at the % of negative vs. positive out of the total non-neutral tweets associated with each candidate

Calculate % positive and negative tweets out of all the non-neutral tweets Framing len vector with number of non-neutral tweets for all candidates Framing dataframe y by combining length vector and matrix x

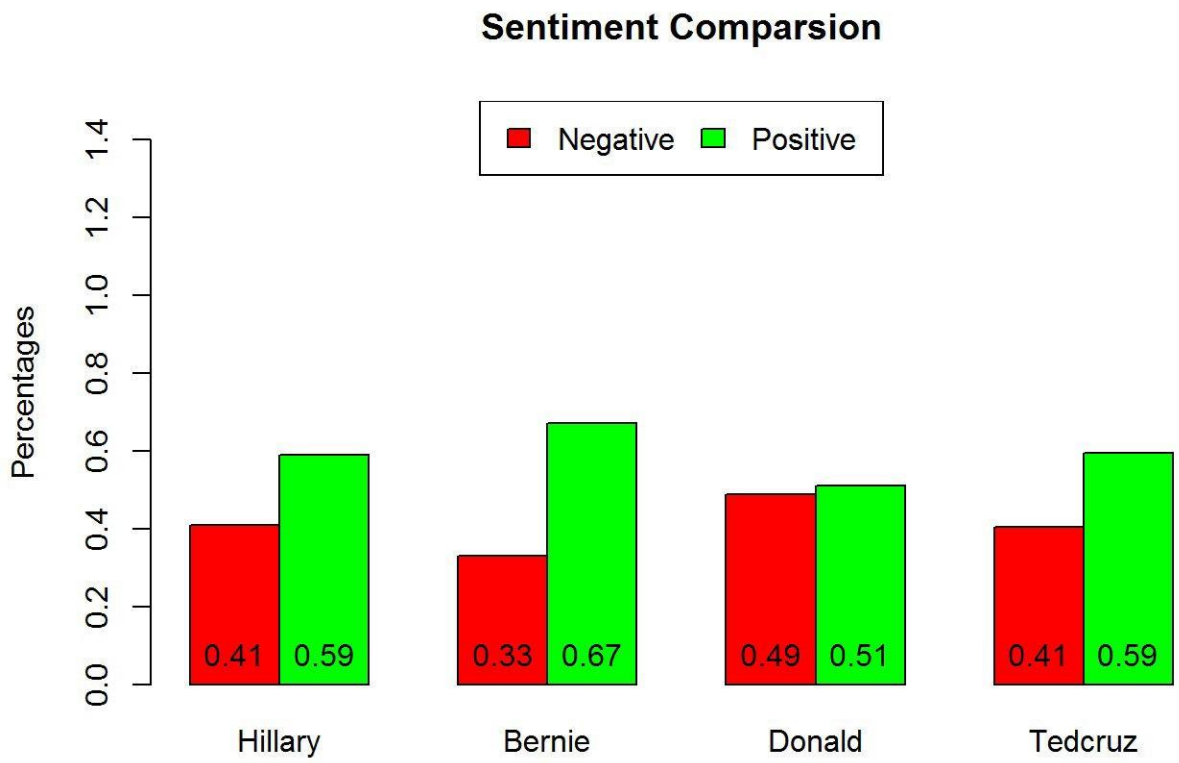
```
len  =  rbind(nrow(Sentiment_scores_hc[Sentiment_scores_hc$score!=0,]),nrow(Senti
ment_scores_hc[Sentiment_scores_bs$score!=0,]),nrow(Sentiment_scores_hc[Sentime
nt_scores_dt$score!=0,]),nrow(Sentiment_scores_hc[Sentiment_scores_tc$score!=
0,]))
y=  cbind.data.frame(x,len)
```

Divide the number of positive and negative score by total number of non-neutral tweets converting and sub setting to get percentages of negative and positive tweets

```
y$Negative  =  y$Negative/y$len
y$Positive  =  y$Positive/y$len
y=  y[,c(1,3)]
y  =  t(y)
```

Bar plot to see what % of tweets are positive vs. negative for each candidate

```
colors = c("red",  "green")
bp  =  barplot(y,beside=T,  col=colors,ylim=c(0,1.5),ylab="Percentages",main="Sent
iment  Comparsion")
text(bp,  0,  round(y,2),cex=1,pos=3)
legend("top",legend=rownames(y),horiz  =  TRUE,fill=colors)
```



Twitter has evolved as source of information for analysts and stakeholders in variety of industries. The nature of these microblogs are very specific and opinionated which makes it more valuable as source of public sentiment data relate to current issues. The results from this project can provide insights to both analysts and candidates about the general public notion in a very real time manner.

D: Topic modeling

Latent Dirichlet Algorithm (LDA) will be used to perform Topic Modeling. LDA assumes each document consists of topics. For example, observe the below tweets:

1. Hillary stands 4 welcoming immigration, economic opportunity & religious tolerance! #IllMarchForThat #NevadaCaucus
2. Latinos in Nevada Cheer #Hillary Clintons Wonky Policy Details on Immigration Reform
3. I'm sure Hillary has disavowed the militant group "Black Lives Matter", correct? Oh, the media hasn't asked her to? #Shocker #TrumpTrain

4. How Hillary Clinton won the battle for the black vote in South Carolina
5. Clinton vows to move on immigration but told BLM to wait #FeelTheBern #Hillary2016 #Hillary #BLM #BlackLivesMatter

LDA Topics

Topic A

40% Immigration, 20% Nevada, 20% Latinos…something to do with immigration and Nevada
Topic B
40% Black, 20% South, 20% Carolina, 10%Lives.something to do with black vote and South Carolina
Documents

1. 100% Topic A
2. 100% Topic A
3. 100% Topic B
4. 100% Topic B
5. 40% Topic A, 60% Topic B

A document will have probability for each topic.
A brief description of LDA:
The document term matrix is given below.

	black	lives	matter	south	carolina	immigration	lations	nevada
Document 1	0	0	1	0	0	1	1	1
Document 2	0	0	0	0	1	2	0	1
Document 3	0	0	0	1	0	1	1	2
Document 4	5	0	2	0	0	1	1	1
Document 5	0	0	0	0	1	2	1	1
Document 6	2	5	3	1	2	0	0	0
Document 7	1	2	0	1	3	1	2	1
Document 8	5	1	1	3	1	0	1	0
Document 9	4	7	2	2	1	0	0	0
Document 10	2	4	2	0	2	0	1	0

For illustration, we make 2 assumptions to start with, that there are 2 topics and 10 tweets are from a vocabulary of 8 words. But, we have no information about how these words are associated to each topic and how documents are associated with topics. So, we make an assumption about the association of documents with a particular topic and then update it with latest information. For example, in the above example we assume that top 5 documents is of Topic A and next 5 documents is of Topic B. So our assignment of topics would be (A, A, A, A, A, B, B, B, B, B) for each document.

Now, we update our assumption based on this move. Assuming this correct, now would we modify our beliefs about topic one? Glancing at the table, I can say at least that the word “black” should be more strongly associated with topic two than with topic one since it occurs in all of the 5 articles (100%) assigned to topic two whereas it occurs in only 1 of 5 articles (20%) assigned to topic one

The second move takes this information and re-assigns the documents to topics. We can observe that Document 4 has more frequency for word ‘black’, so it the probability will be higher that this came from Topic 2. This would change our topic assignment vector to (A, A, A, B, A, B, B, B, B, B). We take each article in turn and guess a new topic assignment and accordingly change the assignment of terms to topics.

This main mathematical concept it uses is conditional probability p (B|A) i.e., the chance of occurrence of B given A has already occurred as illustrated above.

Implementation in R

Take tweets with at least 4 words to improve the relevance and accuracy of LDA analysis. Change the document term matrix to matrix and filtering out tweets with less than or equal to 4 words.

First line of code which is commented below should be uncommented when executing the code, this code is commented for the purpose of report generation.

```
#topic_dtm_hc = DocumentTermMatrix(hc)
n=as.matrix(topic_dtm_hc)
n=n[rowSums(n, na.rm = FALSE)>4,]
```

LDA algorithm is used on matrix using ‘Gibbs’ method

burnin - Gibbs sampling works on Random walk technique(i.e. A random walk is a mathematical formalization of a path that consists of a succession of random steps). In Random walk, starting point of the walk will be chosen at random, it is necessary to discard the first few steps of the walk as those steps will not reflect the actual distribution. This is referred to as the burn-in period.

iter - Iter refers to iteration. In order to avoid correlations between samples we are performing 1000 iterations.

thin - Of the 1000 iterations, we take every 50th iteration for further use
nstart - We use different runs at independent start points
seed - In order to reproduce the same sample

best - best is set to True, to instruct the LDA algorithm to return model with higher probability

```
k<-2
ldaOut <-LDA(n,k, method="Gibbs", control=list(nstart=5, seed =list(200,5,654,9
87,4567) , best=TRUE, burnin =1000, iter =1000, thin=50))
```

Below code is used to get a matrix of document by associated topic

```
#write out results
ldatopics <- as.matrix(topics(ldaOut))
row.names(ldatopics) <- paste("Document",c(1:nrow(n)))
colnames(ldatopics) <- "Topic"
write.csv(ldatopics,"DocsToTopics.csv")
```

```
head(ldatopics)
```

```
##            Topic
## Document 1     1
## Document 2     1
## Document 3     1
## Document 4     2
## Document 5     1
## Document 6     2
```

This block of code returns the topics and terms under it, we can restrict the number of terms here we should specify the number of terms that we want under each topic. In the below code we used '6' which gives 6 terms under each topic

```
ldaterms <- as.matrix (terms(ldaOut,6))
write.csv (ldaterms,"TopicsToTerms.csv")
```

Execute ldaterms object gives the topics and the terms under each topic

```
ldaterms
```

```
##      Topic 1   Topic 2
## [1,] "immigr"  "black"
## [2,] "rubio"   "vote"
## [3,] "support" "get"
## [4,] "sander"  "voter"
## [5,] "reform"  "peopl"
## [6,] "obama"   "america"
```

Topic probabilities are calculated using below code

```
topicProb <- as.data.frame(ldaOut@gamma)
colnames(topicProb) <- c("Topic 1","Topic 2")
write.csv(topicProb,"TopicProbabilities.csv")
```

Output of data frame up to few rows

```
head(topicProb)
```

```
##      Topic 1   Topic 2
## 1 0.5263158 0.4736842
## 2 0.5254237 0.4745763
## 3 0.5254237 0.4745763
## 4 0.4736842 0.5263158
## 5 0.5423729 0.4576271
## 6 0.4909091 0.5090909
```

8: Using Correlated Topic Models (CTM)

CTM approach helps to identify the presence of correlation of one latent topic with another topic. For example, a document about the topic “semantic web” is more likely to be also about the topic “information retrieval” than about the topic “genetics”.

Method used for fitting the models is Variational Expectation Maximization (VEM)

```
ctmout<- CTM(n,k, method="VEM",control = NULL,model = NULL )
```

Converting the output of CTM function to a matrix

```
ctmtopics <- as.matrix(topics(ctmout))
row.names(ctmtopics) <- paste("Document",c(1:nrow(n)))
colnames(ctmtopics) <- "Topic"
write.csv(ctmtopics,"CTMDocsToTopics.csv")
```

Output of data frame up to few rows

```
head(ctmtopics)
```

```
##            Topic
## Document 1     2
## Document 2     1
## Document 3     2
## Document 4     2
## Document 5     2
## Document 6     2
```

This block of code returns the topics and terms under it, we can restrict the number of terms here we should specify the number of terms that we want under each topic Eg: In the below code we used ‘6’ which gives 6 terms under each topic

```
ctmterms <- as.matrix(terms(ctmout,6))
write.csv(ctmterms,"CTMTopicsToTerms.csv")
```

Executing below object gives the topics and the terms under each topic

```
ctmterms
```

```
##      Topic 1   Topic 2
## [1,] "immigr"  "black"
## [2,] "black"   "get"
## [3,] "vote"    "voter"
## [4,] "rubio"   "reform"
## [5,] "america" "peopl"
## [6,] "hillari" "sander"
```

Calculating topic probabilities

```
ctmtopicProb <- as.data.frame(ctmout@gamma)
colnames(ctmtopicProb) <- c("Topic 1","Topic 2")
write.csv(ctmtopicProb,"CTMTopicProbabilities.csv")
```

Shows the output of dataframe up to few rows

```
head(ctmtopicProb)
```

```
##     Topic 1   Topic 2
## 1 0.4960903 0.5039097
## 2 0.5247939 0.4752061
## 3 0.4796378 0.5203622
## 4 0.4770772 0.5229228
## 5 0.4946644 0.5053356
## 6 0.4896649 0.5103351
```