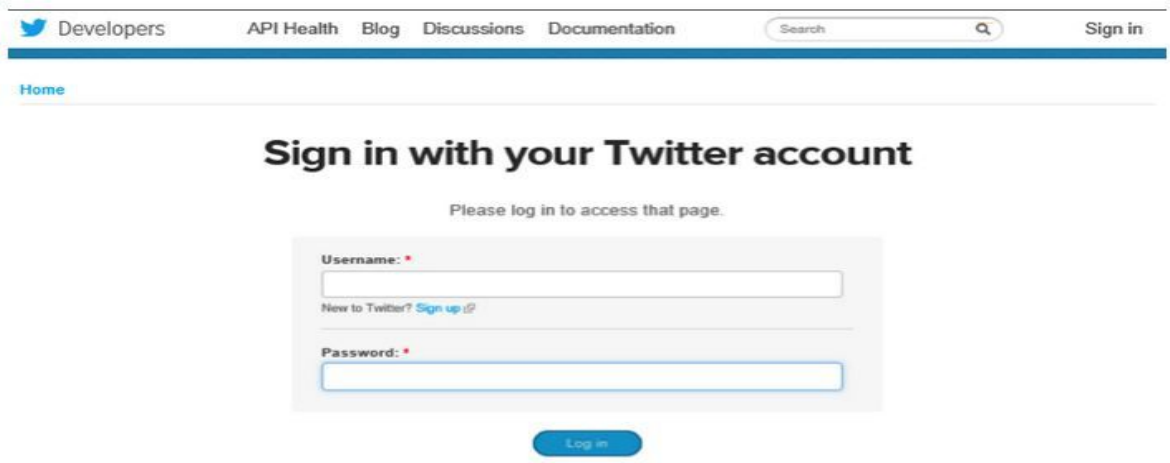
Twitter sentiment analysis using R

Step 1: Twitter Authentication for extracting tweets

Twitter is a popular social platform for expressing our emotions, activities and also for getting a massive amount of information around the web.

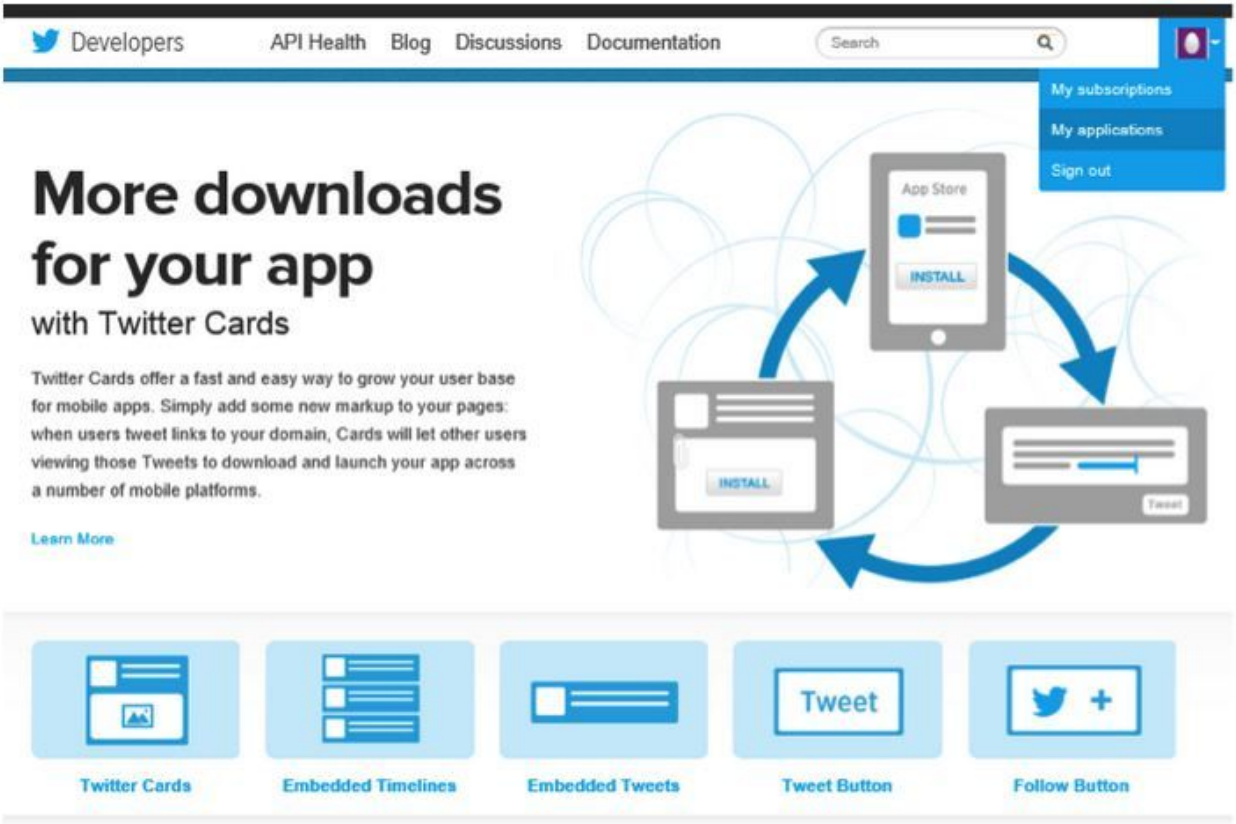
Creating a Twitter Application:

1. First step we need to create Twitter Application (https://apps.twitter.com/) in order to have an access to Twitter's API. Then we will get Consumer Key and Consumer Secret. This application will allow you to perform analysis by connecting your R console to the twitter using the Twitter API.

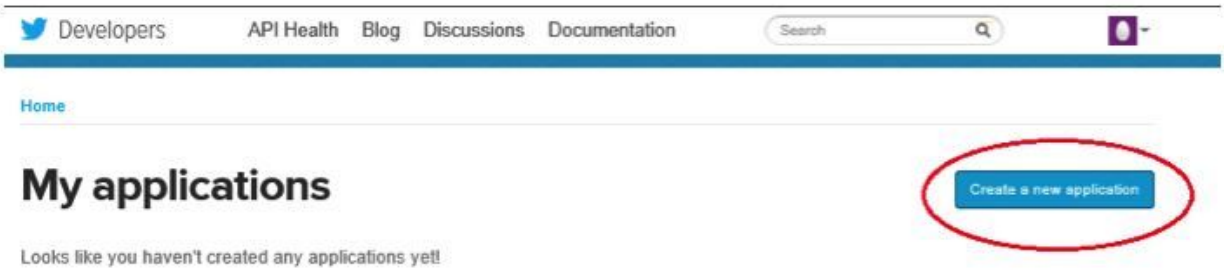


The steps for creating your twitter applications are:

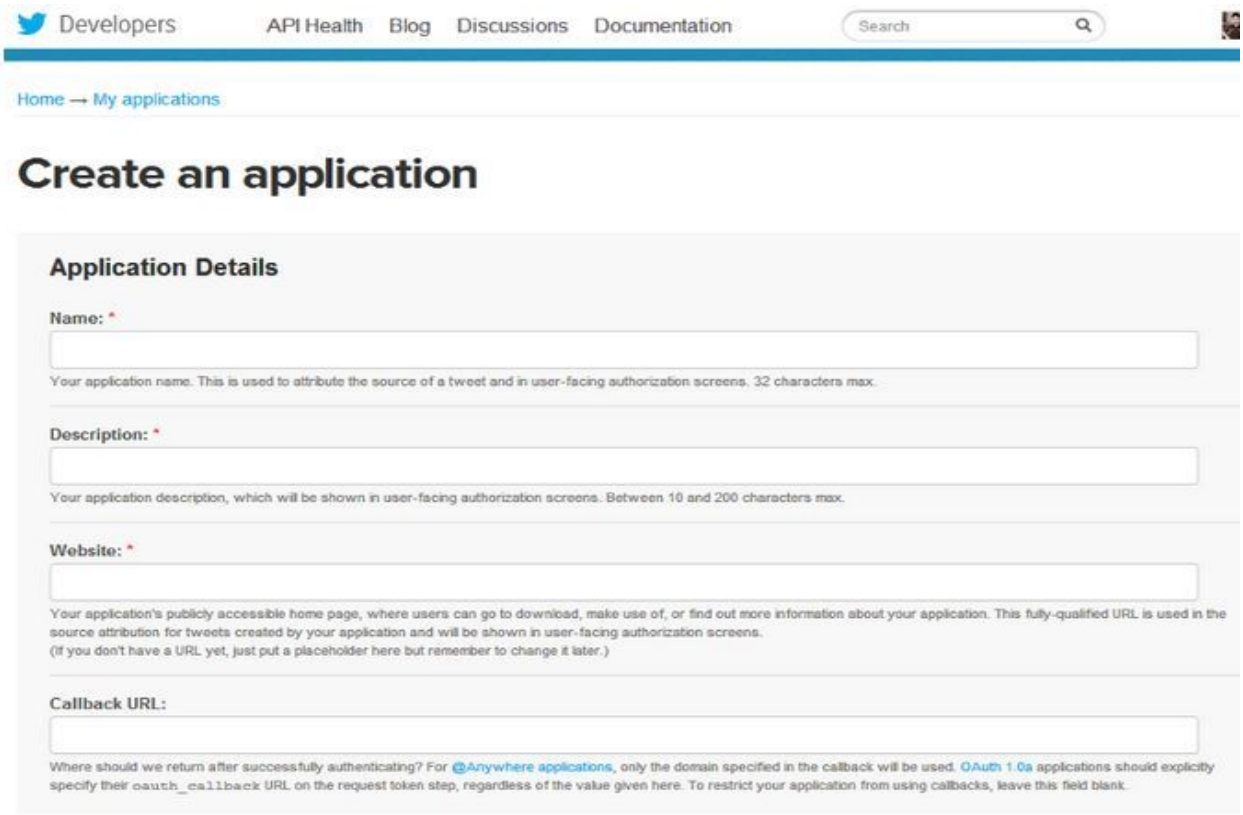
2. Now you can see your Profile picture in the upper right corner and a drop-down menu. In this menu you can find "**My Applications**". Navigate to "**My Applications**" in the upper right hand corner.



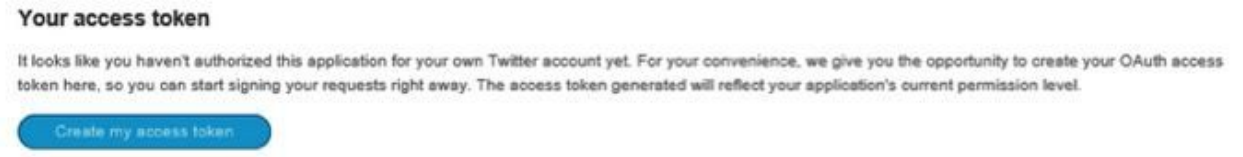
3. Click on Create a new application.



4. Give your application a name. Give a description for application in few words, provide your website's URL or your blog address . Leave the Callback URL blank for now. Complete other formalities and create your twitter application.



5. Scroll down and click on “Create my access token” button.



We have a screen with all the OAuth setting of your new App. Just leave this window in the background; we´ll need it later.
Please note the Consumer key , Consumer Secret, Access Token and Access Token Secret numbers as they will be used in R later.

Once the Twitter Application is ready we can now move forward towards programming in R to extract data from Twitter.

Step 2. Install and Load R Packages

Open your R console and start by loading the following libraries.

```
#connect all libraries (Install and load libraries)
# I have install only one here for understanding how to install packages in R do same for others
install.packages("stringr")

#Load packages(libraries)
library('stringr')              #Fast and friendly string manipulation
library('readr')
library('wordcloud')
library('tm')
library('SnowballC')
library('RWeka')
library('RSentiment')
library(DT)
library(sentimentr)
library(syuzhet)
library(twitteR)            #Provides an interface to the Twitter web API
library(plyr)
library(janeaustenr)
library(tidytext)
library(tm)
library(data.table)
library(httr)
library(twitteR)
library(RColorBrewer)
library(plyr)
library(ggplot2)
library(RCurl)
library(purrr)
library(dplyr)
library(RCurl)
library(streamR)
library(lubridate)

########.....sentiment.....###############
#install sentiment library
library(devtools)
#install_github('sentiment140', 'okugami79')
library(sentiment)
sentiment('I LOVE #Apple')
```

Step 3: Authorization for the Twitter account

In the **consumerKey** field paste the access token you got for your twitter developer application.
In the **consumerSecret** field paste the access token you got for your twitter developer application.
In the **accessToken** field paste the access token you got for your twitter developer application.
In the **accessTokenSecret** field paste the access token you got for your twitter developer application.

```
# Find OAuth settings for twitter:
oauth_endpoints('twitter')

api_key <- 'API Key'
api_secret <- 'API Secret'
access_token <- 'Access token'
access_token_secret <- 'Access token secret'

requestURL= 'https://api.twitter.com/oauth/request_token'
accessURL= 'https://api.twitter.com/oauth/access_token'
authURL= 'https://api.twitter.com/oauth/authorize'

#create an object "cred" that will save the authenticated object that we can use for later
#sessions
cred <- OAuthFactory$new(consumerKey=api_key,
                         consumerSecret=api_secret ,
                         requestURL='https://api.twitter.com/oauth/request_token',
                         accessURL='https://api.twitter.com/oauth/access_token',
                         authURL='https://api.twitter.com/oauth/authorize')

cred$handshake()

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

Step 4. Extract Tweets using R

Now we are ready to extract tweets form Twitter .We set two variables, one for the search string, which could be a hashtag or user mention, and the second variable is the number of tweets we want to extract for analysis

```
#Extract tweets from Twitter
tweets = userTimeline('@MayoClinic', n=1000)
tweets
```

Step 5:Sentiment Analysis

Now we will do Sentiment analysis on the tweets to understand users reactions.

What is Sentiment Analysis?

Sentiment Analysis is the process of determining whether a **piece of writing is positive, negative or neutral**.
We will classify the sentiment of a tweet based on the polarity of the individual words. Each word will be given a score of **+1** if classified as **positive**, **-1** if **negative**, and **0** if classified as **neutral**.
First we need to download the Positive and negative words text files and upload it into the R console.

1) Method 1: Lexicon -using own dictionary of positive and negative files

Step a : Extract tweets and convert it into data frame

```
list <- userTimeline('@MayoClinic', n=5000)
df <- twListToDF(list)
df <- df[, order(names(df))]
df$created <- strftime(df$created, '%Y-%m-%d')
```

Step b : load the word files into R

```
#import pos and neg word files
pos.words=readLines("C:/Users/shwetag/Downloads/positive_words.txt")
neg.words=readLines("C:/Users/shwetag/Downloads/negative_words.txt")
```

Step c :add positive and negative words

```
pos = c(pos.words,'friendly','best','great','focus','Experts','comfortable','highly','thank
you','thanks','available','safely','Inspired','everyone','very','quick','staff','excellent')
neg =
c(neg.words,'Incompetent','Told','Refuse','Claim','Bill','Paid','Money','stress','Rude','wtf','behind','f
eels', 'ugly', 'back','worse' ,'bad', 'no','freaking','sucks','horrible')
```

Step d : Sentiment Function

Once we have the tweets we just need to apply some functions to convert these tweets into some useful information. The main working principle of sentiment analysis is to find the words in the tweets that represent positive sentiments and find the words in the tweets that represent negative sentiments.

```
#Define  function score.sentiment

score.sentiment = function(sentences, pos.words, neg.words, .progress='none')
{
  # Parameters
  # sentences: vector of text to score
  # pos.words: vector of words of postive sentiment
  # neg.words: vector of words of negative sentiment
  # .progress: passed to laply() to control of progress bar

# create simple array of scores with laply
scores = laply(sentences,
               function(sentence, pos.words, neg.words)
               {
                 # remove punctuation
                 sentence = gsub("[[:punct:]]", "", sentence)
                 # remove control characters
                 sentence = gsub("[[:cntrl:]]", "", sentence)
                 # remove digits?
                 sentence = gsub('\\d+', '', sentence)

                 # define error handling function when trying tolower
                 tryTolower = function(x)
                 {
                   # create missing value
                   y = NA
                   # tryCatch error
                   try_error = tryCatch(tolower(x), error=function(e) e)
                   # if not an error
                   if (!inherits(try_error, "error"))
                     y = tolower(x)
                   # result
                   return(y)
                 }
                 # use tryTolower with sapply
                 sentence = sapply(sentence, tryTolower)

                 # split sentence into words with str_split (stringr package)
                 word.list = str_split(sentence, "\\s+")
                 words = unlist(word.list)

                 # compare words to the dictionaries of positive & negative terms
                 pos.matches = match(words, pos.words)
```

```
        neg.matches = match(words, neg.words)

        # get the position of the matched term or NA
        # we just want a TRUE/FALSE
        pos.matches = !is.na(pos.matches)
        neg.matches = !is.na(neg.matches)

        # final score
        score = sum(pos.matches) - sum(neg.matches)
        return(score)
      }, pos.words, neg.words, .progress=.progress )

  # data frame with scores for each sentence
  scores.df = data.frame(text=sentences, score=scores)
  return(scores.df)
```

Step e : Apply sentiment Function to the tweets

```
#find out score
result <- score.sentiment(df$text,pos,neg)
head(result$score,15)
 [1]  1 -1 -2  0 -1  0  0  1  0  1  2  0  1  0  1

dim(result)
[1] 774   2

stat <- result['score']
stat$created <- df$created
table(result$score)

 -5  -3  -2  -1   0   1   2   3   4
  1   8  39 127 328 195  58  16   2

#Labelling positive negative and neutral
stat <- mutate(stat, tweet=ifelse(stat$score > 0, 'positive', ifelse(stat$score < 0, 'negative',
'neutral')))
by.tweet <- group_by(stat, tweet, created)
table(by.tweet$tweet)
```

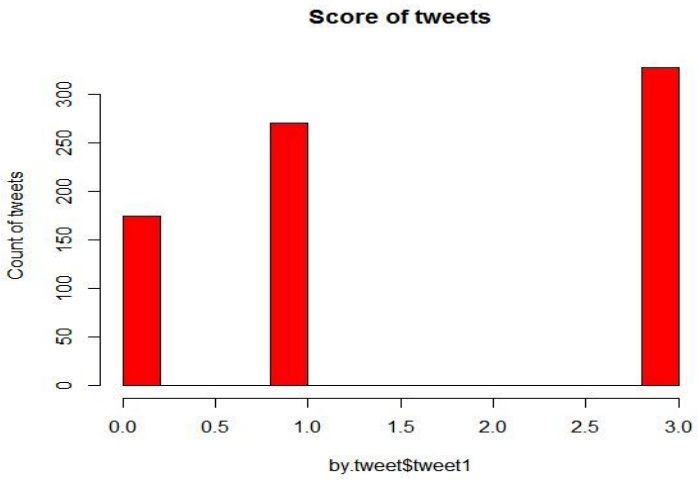
negative	**neutral**	**positive**
175	**328**	**271**

prop.table(table(by.tweet$tweet))

negative	neutral	positive
0.2260982	0.4237726	0.3501292

Step f : Histogram of the Scores

by.tweet$tweet1=ifelse(by.tweet$tweet== 'positive',1 , ifelse(by.tweet$tweet== 'negative',0,3))
hist(by.tweet$tweet1, col='red',main ='Score of tweets', ylab = 'Count of tweets')



Here positive=1, Negative=0, Neutral=3

Per Day sentiment Analysis:

stat1 <- result['score']
stat1$created <- df$created
#convert date column into specific formate(yy-mm-dd)
date.vec = paste(stat1$created)
stat1$date = as.Date(strptime(date.vec, "%Y-%m-%d"))
stat1=stat1[,c(3,1)]
stat1$date[order(stat1$date , decreasing=T)]

#summarize by date and number of tweets
detach("package:plyr", unload=TRUE)
a1=stat1 %>% group_by(date=floor_date(date, "day")) %>% summarize(number=n(),
score=sum(score))
#Labelling positive negative and neutral
a1 <- mutate(a1, tweet=ifelse(a1$score > 0, 'positive', ifelse(a1$score < 0, 'negative', 'neutral')))
by.tweet1 <- group_by(a1, tweet, date)

by.tweet1

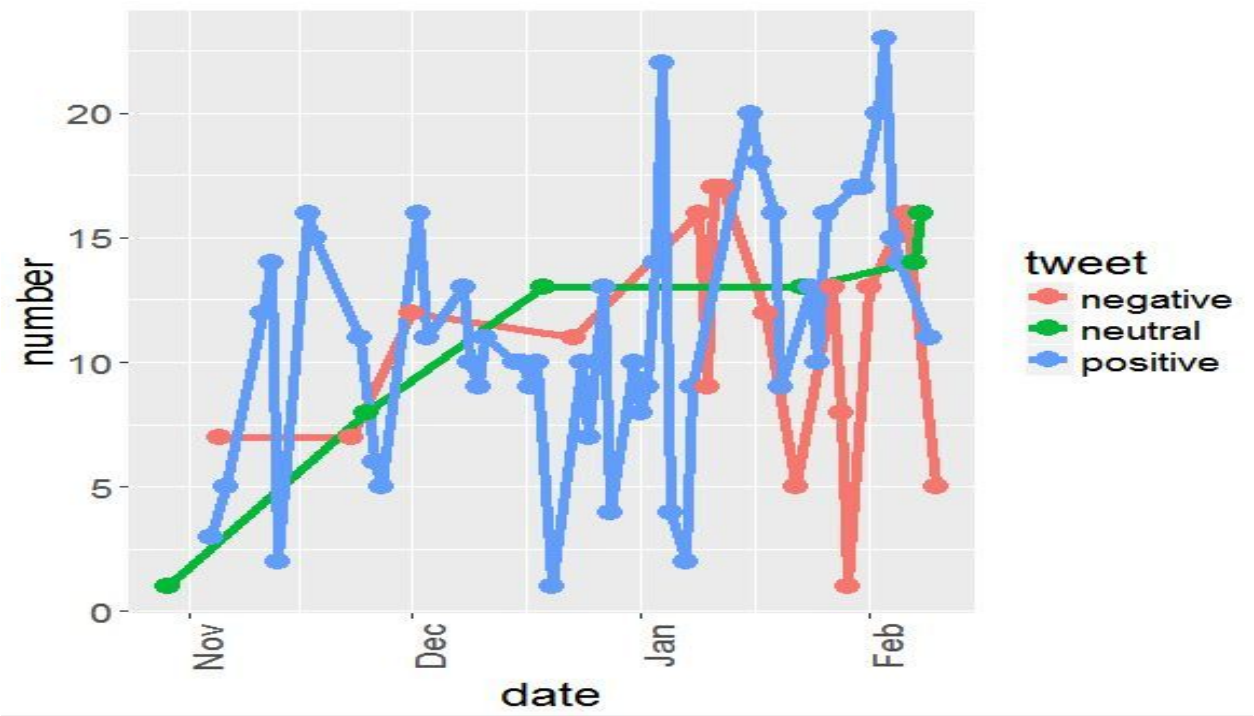
```
> by.tweet1
# A tibble: 70 x 4
# Groups:   tweet, date [70]
   date       number score tweet
   <date>      <int> <int> <chr>
 1 2017-10-29      1     0 neutral
 2 2017-11-04      3     1 positive
 3 2017-11-05      7    -4 negative
 4 2017-11-06      5     3 positive
 5 2017-11-11     12     3 positive
 6 2017-11-12     14     3 positive
 7 2017-11-13      2     1 positive
 8 2017-11-17     16     9 positive
 9 2017-11-18     15     1 positive
10 2017-11-23      7    -2 negative
# ... with 60 more rows
```

table(by.tweet1$tweet)

negative neutral positive
16 6 48

#Graphical representation

```
ggplot(by.tweet1, aes(date, number)) + geom_line(aes(group=tweet, color=tweet), size=2) +
  geom_point(aes(group=tweet, color=tweet), size=4) +
  theme(text = element_text(size=18), axis.text.x = element_text(angle=90, vjust=1))
```



2)TF_IDF

Suppose you want to summarize a document or a paragraph using few keywords.
TF Score (Term Frequency): One technique is to pick the most frequently occurring terms (words with high term frequency or tf). However, the most frequent word is a less useful metric since some words like 'this', 'a' occur very frequently across all documents.
IDF Score (Inverse Document Frequency): Hence, we also want a measure of how unique a word is i.e. how infrequently the word occurs across all documents (inverse document frequency or idf).
tfidf: Hence, the product of tf x idf (tfidf) of a word gives a product of how frequent this word is in the document multiplied by how unique the word is w.r.t. the entire corpus of documents.
Words in the document with a high tfidf score occur frequently in the document and provide the most information about that specific document.
The tf.idf score increases with number of occurrences within a document.

This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.Variations of the tf-idf weighting scheme is a central tool in scoring and ranking a **document's relevance** given a user query.

 # Function for data cleaning

```
f_clean_tweets <- function (tweets) {

  clean_tweets = sapply(tweets, function(x) x$getText())
  # remove retweet entities
  clean_tweets = gsub('(RT|via)((?:\\b\\W*@\\w+)+)', '', clean_tweets)
  # remove at people
  clean_tweets = gsub('@\\w+', '', clean_tweets)
  # remove punctuation
  clean_tweets = gsub('[[:punct:]]', '', clean_tweets)
  # remove numbers
  clean_tweets = gsub('[[:digit:]]', '', clean_tweets)
  # remove html links
  clean_tweets = gsub('http\\w+', '', clean_tweets)
  # remove unnecessary spaces
  clean_tweets = gsub('[ \t]{2,}', '', clean_tweets)
  clean_tweets = gsub('^\\s+|\\s+$', '', clean_tweets)
  # remove emojis or special characters
  clean_tweets = gsub('<.*>', '', enc2native(clean_tweets))
```

```
  clean_tweets = tolower(clean_tweets)
  clean_tweets
}

#call the function defined above to clean the data
text <- f_clean_tweets(tweets)
a=length(text)
text_df <- data_frame(line = 1:a, text = text)

#tokenize
tidy_data<-text_df %>%
  unnest_tokens(word,text)

#removing Stopwords
data("stop_words")

#data with count
tidy_data %>%
  count(line, word, sort = T)
tidy_data <- tidy_data%>%
  anti_join(stop_words)

book_words <- tidy_data %>%
  count(line, word, sort = TRUE) %>%
  ungroup()

#total count of word in a document
total_words <- book_words %>%
  group_by(line) %>%
  summarize(total = sum(n))
book_words <- left_join(book_words, total_words)
book_words
```

```
> book_words
# A tibble: 3,177 x 4
    line word          n total
   <int> <chr>     <int> <int>
 1   214 coconut       3    11
 2     7 heart         2     9
 3    17 women         2    10
 4    32 stretching    2     9
 5    33 eat           2     7
 6    37 faqs          2    11
 7    37 flu           2    11
 8    68 nodules       2     9
 9    80 awareness     2    10
10    96 dr            2    13
# ... with 3,167 more rows
```

#term frequency

```
freq_by_rank <- book_words %>%
  group_by(line) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)
```

 # tf_idf

```
book_words <- book_words %>%
  bind_tf_idf(word, line, n)
```

```
# A tibble: 3,177 x 7
    line word           n total    tf   idf tf_idf
   <int> <chr>      <int> <int> <dbl> <dbl>  <dbl>
 1   214 coconut        3    11 0.273  6.00  1.64
 2     7 heart          2     9 0.222  3.36  0.748
 3    17 women          2    10 0.200  4.39  0.879
 4    32 stretching     2     9 0.222  6.00  1.33
 5    33 eat            2     7 0.286  3.81  1.09
 6    37 faqs           2    11 0.182  5.31  0.966
 7    37 flu            2    11 0.182  3.30  0.599
 8    68 nodules        2     9 0.222  6.00  1.33
 9    80 awareness      2    10 0.200  4.39  0.879
10    96 dr             2    13 0.154  3.01  0.463
# ... with 3,167 more rows
```

#tf_idf in descending order

```
book_words %>%
  arrange(desc(tf_idf))
```

```
# A tibble: 3,177 x 7
    line word           n total    tf   idf tf_idf
   <int> <chr>      <int> <int> <dbl> <dbl>  <dbl>
 1   210 geneticte…     1     1 1.00   6.00   6.00
 2   218 thyroidca…     1     1 1.00   6.00   6.00
 3    87 caffeine       1     2 0.500  6.00   3.00
 4   100 aarticle       1     2 0.500  6.00   3.00
 5   100 happening      1     2 0.500  6.00   3.00
 6   130 hangnail       1     2 0.500  6.00   3.00
 7   130 rip            1     2 0.500  6.00   3.00
 8   331 attention      1     2 0.500  6.00   3.00
 9   331 bringing       1     2 0.500  6.00   3.00
10   361 anesthesia     1     2 0.500  6.00   3.00
# ... with 3,167 more rows
```

Here we can say that tweet no. 210 is more relevant to tweet no. 218 and likewise.

Method2) NRC lexicon method

```
tweetSentiments1 <- get_nrc_sentiment(df$text)
tweets1 <- cbind(tweets.df, tweetSentiments1)
dim(tweets1)
sentimentTotals <- data.frame(colSums(tweets1[,c(17:26)]))
```

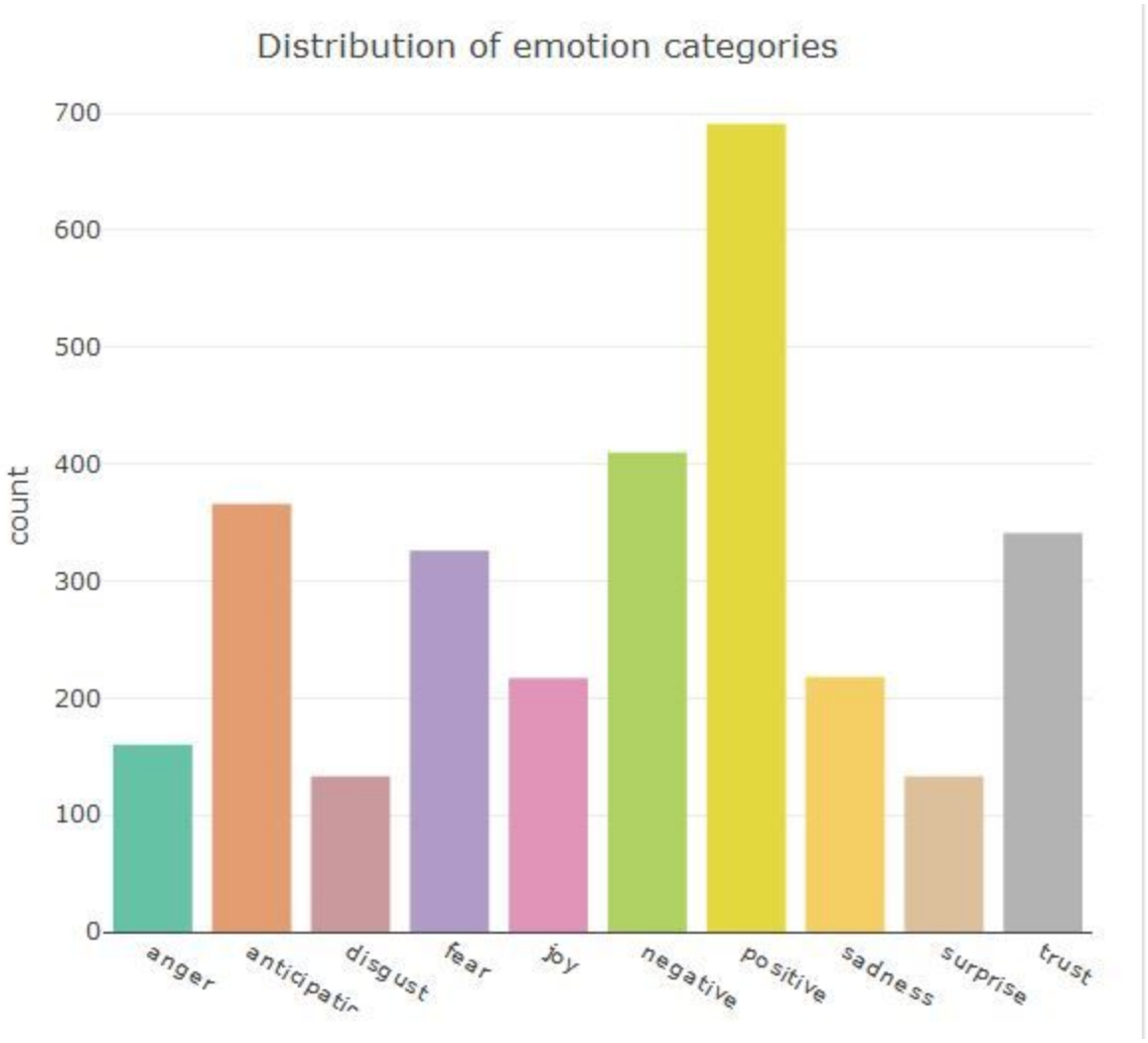
```
> head(tweets1[,c(1,17:26)])
    created anger anticipation disgust fear joy sadness surprise trust negative positive
1 2018-02-06     0            1       0    1       0     1        0            1        1        1
2 2018-02-06     0            0       0    0       0     0        0            0        0        0
3 2018-02-06     0            1       0    1       0     0        0            0        1        0
4 2018-02-06     1            1       0    1       0     1        0            0        2        0
5 2018-02-06     1            2       2    2       1     2        2            1        2        3
6 2018-02-06     0            0       0    0       0     0        0            0        0        2
```

```
names(sentimentTotals) <- "count"
sentimentTotals <- cbind("sentiment" = rownames(sentimentTotals), sentimentTotals)
rownames(sentimentTotals) <- NULL
```

```
sentimentTotals
```

	sentiment	count
1	anger	160
2	anticipation	366
3	disgust	133
4	fear	326
5	joy	217
6	sadness	218
7	surprise	133
8	trust	341
9	negative	410
10	positive	691

```
# Visualize the emotions from NRC sentiments
library(plotly)
plot_ly(sentimentTotals, x=~sentiment, y=~count, type='bar', color=~sentiment) %>%
  layout(xaxis=list(title=''), showlegend=FALSE,
         title='Distribution of emotion categories')
```


Distribution of emotion categories
count
700
600
500
400
300
200
100
0
anger
anticipatio
disgust
fear
joy
negative
positive
sadness
surprise
trust

Method 3) Machine learning
1)Naive bayes

```
list <- userTimeline('@MayoClinic', n=1000)
df <- twListToDF(list)
#find out score
result <- score.sentiment(df$text,pos,neg)
stat=result$score
#Labelling positive negative and neutral
tweet=ifelse(stat > 0, 'positive', ifelse(stat < 0, 'negative', 'neutral'))
class=ifelse(tweet== 'positive', 1 , ifelse(tweet== 'negative',0, 2))
data=data.frame(class, df$text)
dim(data)
```

```
[1] 665   2
```

1.Bag of Words Tokenisation

In this approach, we represent each word in a document as a token (or feature) and each document as a vector of features. In addition, for simplicity, we disregard word order and focus only on the number of occurrences of each word i.e., we represent each document as a multi-set ‘bag’ of words.

We first prepare a corpus of all the documents in the dataframe.

```
corpus <- Corpus(VectorSource(data1$text))
# Inspect the corpus
corpus
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2000
inspect(corpus[1:3])
```

2. Data Cleanup

Next, we clean up the corpus by eliminating numbers, punctuation, white space, and by converting to lower case. In addition, we discard common stop words such as “his”, “our”, “hadn’t”, couldn’t“, etc. We use the tm_map() function from the ‘tm’ package to this end.

```
# Use dplyr's  %>% (pipe) utility to do this neatly.
corpus.clean <- corpus %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeNumbers) %>%
  tm_map(removeWords, stopwords(kind="en")) %>%
  tm_map(stripWhitespace)
```

3. Matrix representation of Bag of Words : The Document Term Matrix

We represent the bag of words tokens with a document term matrix (DTM). The rows of the DTM correspond to documents in the collection, columns correspond to terms, and its elements are the term frequencies. We use a built-in function from the ‘tm’ package to create the DTM.

```
dtm <- DocumentTermMatrix(corpus.clean)
# Inspect the dtm
inspect(dtm[40:50, 10:15])
```

4. Partitioning the Data

Next, we create 75:25 partitions of the dataframe, corpus and document term matrix for training and testing purposes.

```
df.train <- df[1:465,]
df.test <- df[466:665,]

dtm.train <- dtm[1:465,]
dtm.test <- dtm[466:665,]

corpus.clean.train <- corpus.clean[1:465]
corpus.clean.test <- corpus.clean[466:665]

dim(dtm.train)
```

5. Feature Selection

The DTM contains 38957 features but not all of them will be useful for classification. We reduce the number of features by ignoring words which appear in less than five reviews. To do this, we use ‘findFreqTerms’ function to indentify the frequent words, we then restrict the DTM to use only the frequent words using the ‘dictionary’ option.

```
ffreq <- findFreqTerms(dtm.train, 2)
length((ffreq))
## [1] 782

# Use only 5 most frequent words (fivefreq) to build the DTM
```

```
dtm.train.nb <- DocumentTermMatrix(corpus.clean.train, control=list(dictionary = ffreq))

dim(dtm.train.nb)
## [1]  465 782

dtm.test.nb <- DocumentTermMatrix(corpus.clean.test, control=list(dictionary = ffreq))

dim(dtm.test.nb)
```

The Naive Bayes algorithm

The Naive Bayes text classification algorithm is essentially an application of Bayes theorem (with a strong independence assumption) to documents and classes.
9. Boolean feature Multinomial Naive Bayes
We use a variation of the multinomial Naive Bayes algorithm known as binarized (boolean feature) Naive Bayes due to Dan Jurafsky. In this method, the term frequencies are replaced by Boolean presence/absence features. The logic behind this being that for sentiment classification, word occurrence matters more than word frequency.

```
convert_count <- function(x) {
  y <- ifelse(x > 0, 1,0)
  y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
  y
}
# Apply the convert_count function to get final training and testing DTMs
trainNB <- apply(dtm.train.nb, 2, convert_count)
testNB <- apply(dtm.test.nb, 2, convert_count)
```

10. Training the Naive Bayes Model

To train the model we use the naiveBayes function from the ‘e1071’ package. Since Naive Bayes evaluates products of probabilities, we need some way of assigning non-zero probabilities to words which do not occur in the sample. We use Laplace 1 smoothing to this end.

```
# Train the classifier
classifier <- naiveBayes(trainNB, df.train$class, laplace = 1)

# Use the NB classifier we built to make predictions on the test set.
pred <- predict(classifier, newdata=testNB)

# Create a truth table by tabulating the predicted class labels with the actual class labels
t=table("Predictions"= pred,  "Actual" = df.test$class )
```

```
            Prediction

Actual     110     9
            21     52

acc<-sum(diag(t))/sum(t)
acc
[1] 0.84375
```

Accuracy = 0.84375

2)glmnet

```
list <- userTimeline('@MayoClinic', n=1000)
df <- twListToDF(list)
df$sentiment=stat$tweet
head(df)
class=ifelse(df$sentiment== 'positive', 1 , ifelse(df$sentiment== 'negative',0, 2))

data=data.frame(df,df$sentiment,class)
head(data)

# loading packages
library(twitteR)
library(ROAuth)
library(tidyverse)
library(purrrlyr)
library(text2vec)
library(caret)
library(glmnet)
library(ggrepel)
# data splitting on train and test
set.seed(2340)
trainIndex <- createDataPartition(data$sentiment, p = 0.7, list = FALSE, times = 1)
tweets_train <- data[trainIndex, ]
tweets_test <- data[-trainIndex, ]


##### Vectorization #####
# define preprocessing function and tokenization function
```

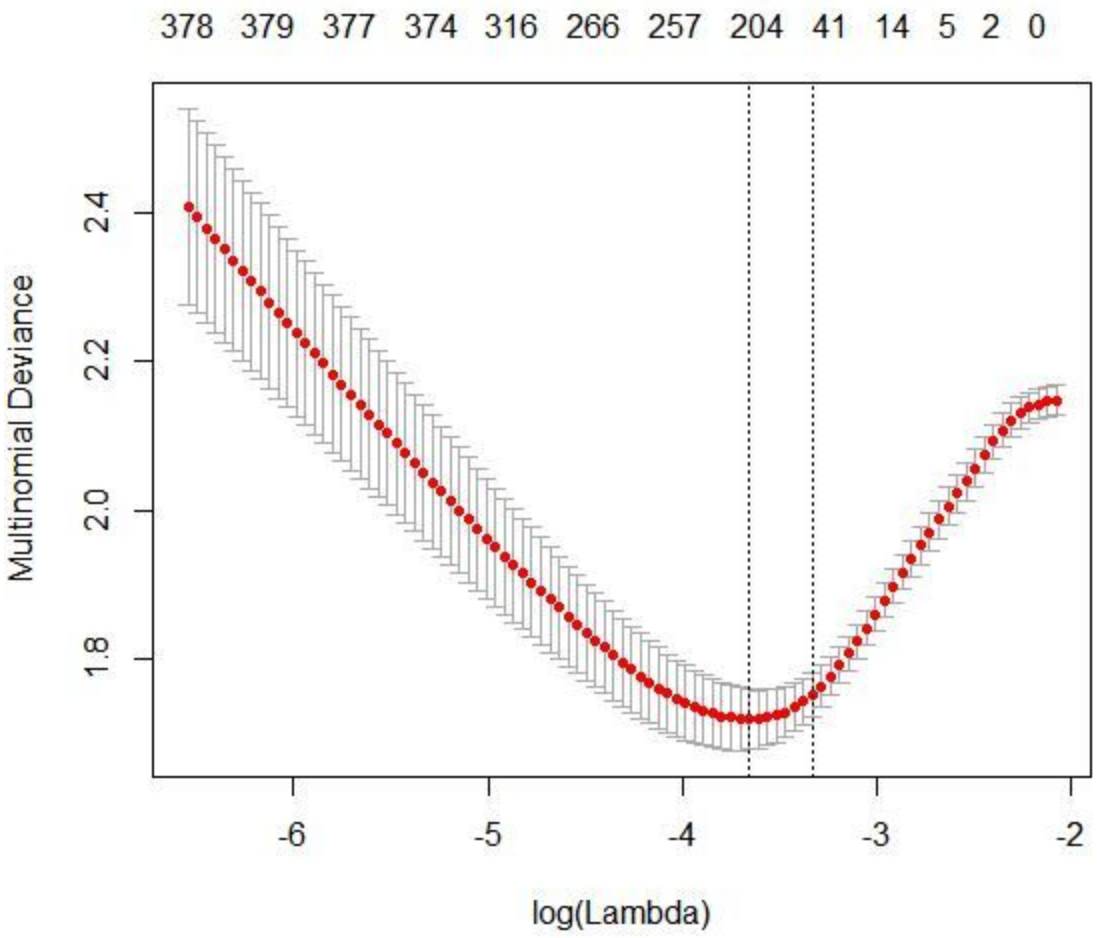
```
prep_fun <- tolower
tok_fun <- word_tokenizer

it_train <- itoken(tweets_train$text,
                   preprocessor = prep_fun,
                   tokenizer = tok_fun,
                   ids = tweets_train$id,
                   progressbar = TRUE)
it_test <- itoken(tweets_test$text,
                  preprocessor = prep_fun,
                  tokenizer = tok_fun,
                  ids = tweets_test$id,
                  progressbar = TRUE)


# creating vocabulary and document-term matrix
vocab <- create_vocabulary(it_train)
vectorizer <- vocab_vectorizer(vocab)
dtm_train <- create_dtm(it_train, vectorizer)
# define tf-idf model
tfidf <- TfIdf$new()
# fit the model to the train data and transform it with the fitted model
dtm_train_tfidf <- fit_transform(dtm_train, tfidf)
# apply pre-trained tf-idf transformation to test data
dtm_test_tfidf  <- create_dtm(it_test, vectorizer) %>%
  transform(tfidf)

# train the model
t1 <- Sys.time()
glmnet_classifier <- cv.glmnet(x = dtm_train_tfidf,
                               y = tweets_train[['sentiment']],
                               family = 'multinomial',
                               # L1 penalty
                               alpha = 1,
                               # interested in the area under ROC curve
                               type.measure = "auc",
                               # 5-fold cross-validation
                               nfolds = 5,
                               # high value is less accurate, but has faster training
                               thresh = 1e-3,
                               # again lower number of iterations for faster training
                               maxit = 1e3)
print(difftime(Sys.time(), t1, units = 'mins'))
```

plot(glmnet_classifier)



Classify Text in Python:

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
```

```
data=pd.read_csv('C:/Users/shwetag/Downloads/a.csv', sep='\t')

data.head()

# Split into Input and Output.
attributes = list(dataset.columns[:2])
X = dataset[attributes].values
y= dataset['class'].values

# Split into train and test sets.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state =0)

# Extracting features from text files
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)


# TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

# Machine Learning
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf,y_train)

# Building a pipeline: We can write less code and do all of the above, by building a pipeline as follows:
# The names 'vect' , 'tfidf' and 'clf' are arbitrary but will be used later.
# We will be using the 'text_clf' going forward.
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])
text_clf = text_clf.fit(X_train, y_train)

# Performance of NB Classifier
import numpy as np
y = fetch_20newsgroups(subset='test', shuffle=True)
predicted = text_clf.predict(y_train)
np.mean(predicted ==y_test)

0.82381837493361654
```