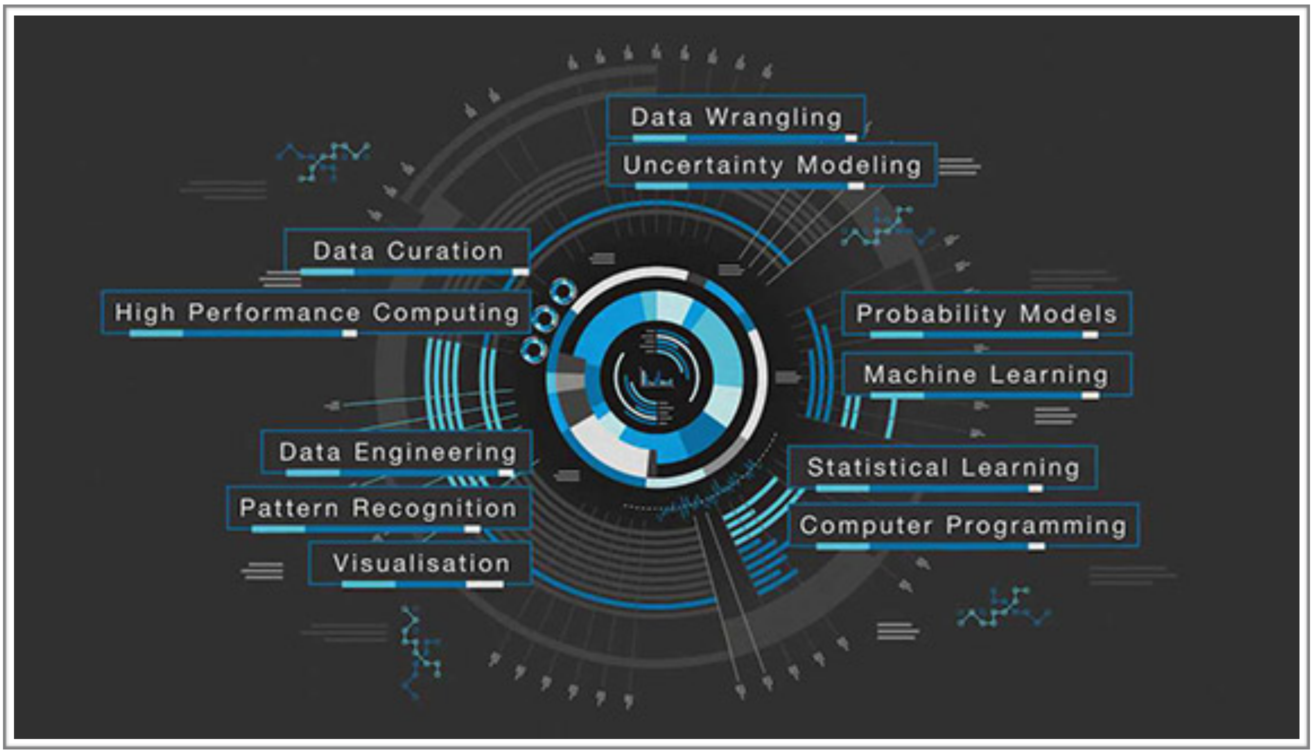
"Summarise Twitter messages related to upcoming Lok Sabha election for predicting the chances of winning party"

Abhishek Hupele

5th-6th April 2019



Contents

1 Introduction

As because of the massive growth of user-created data in the recent WWW websites, people from various backgrounds tweet massive amount of textual remarks deliberating their thoughts in a different perspective of their emotions and public to everyone. Natural Language Processing (NLP) can be categorised into opinion and text mining. This technique is helped for isolating the opinions of posting on various social media platforms like Twitter, Reddit, and Facebook etc. In today's world text or opinion, mining is helpful for judging public views regarding a newly released item, movie, song, book, etc. It also differentiated among positive, negative and neutral opinion and recommendations. It also becomes a common practice for the common people to pose their expressions towards the political leader on social media. Different reporters have been taking an interview with the political leader to know their views and communicate with the people through TV program, YouTube, etc. People express their opinions with each other regarding the political talks which ran on the TV show. It is very expensive and time-consuming task to search people's opinions via surveys and polls. Now various social media sites (like Twitter, Reddit, Facebook, etc.) are extensively used by the public when they can share their opinions publicly. One of important microblogging site receiving around 500 million tweets every day where the daily limit of each user is 2,400 tweets and 140 characters every tweet. Hence, Twitter is one of the relevant platforms where each people can connect with different communities and express their opinions loud and clear.

Sentiment analysis is a type of data mining process that determines the public opinion through NLP. It is the process of classifying opinions into three categories like "positive", or "negative" or "neutral". This data quantifies the public's reactions toward certain people, communities, and political discourses which divulge the contextual polarity of the information.

Due to the second-most populous country and the most populous democracy in the world, India political situation is most fluctuating. Every step of the ruling party would have several views of the oppositions. But today common people post their opinions on the social media regarding every political step (like demonetization of all Indian five hundred and one thousand currency of the Mahatma Gandhi Series).
Therefore our aim is to analyse the emotion of web users concerning every political party, their leaders and their steps based on tweets on the social media.

1.1 Problem statement

Domain	Internal Systems & Defined Problems
Title	Author's Genie: A platform to measure the quality of a course before publication

In today's digital learning platforms, authors typically get to know about the quality of their courses through various online feedbacks once course is live. The idea here is to create a platform which has a model to measure the quality of a given course even before publication so that more effective courses go live.

Hints/suggestions: Quality of a course could be measured on various parameters such as fulfillment of learning objectives, grammar, semantics, look and feel, flow of content, quality of quizzes, and forward and cross references made. NLP and ML libraries have to be used to create this model which has to be embedded in an easy to use interface to load the course content and get easy to understand results on its quality.

Possible Extensions: A few of the extensions to the above problem statement could be:

- Possibly give suggestions on various quality parameters while authoring itself
- May help in self-correction on things such as placement of paras, connector sentences, forward and cross references, etc.

Possible related use cases in Infosys context:

- Measure the quality of proposal documents.
- Measure the quality of client deliverables during projects such as requirement specifications, Architecture document etc.

Use Cases :

Summarise the data set of Twitter messages related to

Use Case 1 : recent upcoming Lok Sabha election, 2019
Use Case 2 : Udemy, Coursera Courses
Use Case 3 : Customer Review : Mobile (Amazon review)

for predicting the chances of winning party by utilising public's opinion.

Possible Extensions :

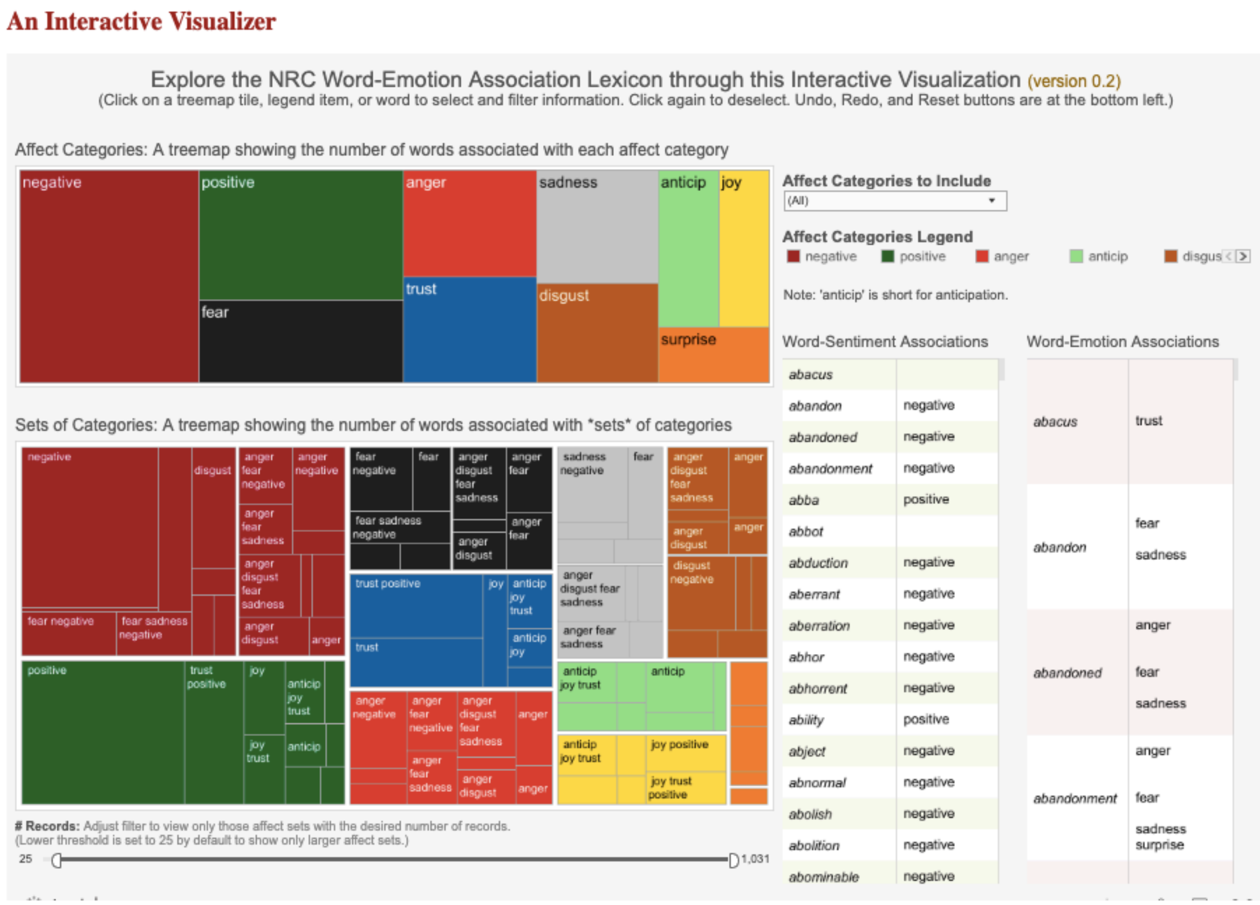
Use Case 1 : recent upcoming Lok Sabha election, 2019

- Trending topics or demands
- Deciding or motivating factors (priority / impact) on vote bank
- Performance of existing government
- Which leaders are popular (ticket allocation)
- Insights Region wise, industry wise etc.
- Election Agenda
- Marketing Campaign
- Sentiment Manipulation

1.2 Data :

Today microblogging has become a very common platform for exchanging opinion among us. Many users exchange their thoughts on a various aspect of their activity. Consequently, microblogging websites are the substantial origin of information for sentiment analysis and opinion mining. Twitter is a famous microblogging website where 500 million tweets are posted every day. This thesis, we summarise the data set of Twitter messages related to upcoming Lok Sabha election, 2019 for predicting the chances of winning party by utilising public's opinion.

We can use NRC Emotion Lexicon to determine the overall tone of the event by eight emotions. Furthermore, we can use a Deep Learning tool named, ParallelDots AI APIs by ParallelDots Inc that can analyse the sentiment into positive, negative and neutral. This tool can be helped to extract various peoples' sentiment and summarise the results for further decision making.



Data Preparation :

The data has to be collected from past months as well as live data using Twitter's streaming API. There are multiple approaches to collect tweets , like

a) Collecting tweets mentioning two verified Twitter accounts named @narendramodi and @RahulGandhi respectively,
b) capture data by the keywords: 'BJP', 'Narendra Modi', 'Rahul Gandhi','INC', 'Indian National Congress','Bhartiya Janta Party'

Capture Live Data :

(i) Getting Data from Twitter Streaming API

API stands for Application Programming Interface. It is a tool that makes the interaction with computer programs and web services easy. Many web services provides APIs to developers to interact with their services and to access data in programmatic way. we will use Twitter Streaming API to download tweets related to keywords.

Step 1: Getting Twitter API keys
In order to access Twitter Streaming API, we need to get 4 pieces of information from Twitter: API key, API secret, Access token and Access token secret. Follow the steps below to get all 4 elements:

- Create a twitter account if you do not already have one.
- Go to https://apps.twitter.com/ and log in with your twitter credentials.
- Click "Create New App"
- Fill out the form, agree to the terms, and click "Create your Twitter application"
- In the next page, click on "API keys" tab, and copy your "API key" and "API secret".
- Scroll down and click "Create my access token", and copy your "Access token" and "Access token secret".

Step 2: Connecting to Twitter Streaming API and downloading data
We will be using a Python library called `Tweepy` to connect to Twitter Streaming API and downloading the data.
Next create, a file called `twitter_live_v1_keyword.py` , and copy into it the code below.

```
from tweepy import Stream
#Variables that contains the user credentials to access Twitter API
access_token = "ENTER YOUR ACCESS TOKEN"
access_token_secret = "ENTER YOUR ACCESS TOKEN SECRET"
consumer_key = "ENTER YOUR API KEY"
consumer_secret = "ENTER YOUR API SECRET"
#This is a basic listener that just prints received tweets to stdout.
class StdOutListener(StreamListener):
   def on_data(self, data):
      print data
return True
   def on_error(self, status):
      print status
   if __name__ == '__main__':

         #This handles Twitter authetification and the connection to Twitter Streaming API
         l = StdOutListener()
         auth = OAuthHandler(consumer_key, consumer_secret)
         auth.set_access_token(access_token, access_token_secret)
         stream = Stream(auth, l)
```

```
#This line filter Twitter Streams to capture data by the keywords: 'INC', 'Indian National
#Congress','Rahul Gandhi'
stream.filter(track=[''INC', 'Indian National Congress','Rahul Gandhi'])
```

If we run the program from our terminal using the command: `python twitter_live_v1_keyword.py` , we will see data flowing like the picture below.


twitter_live_v1_congress_5apr2300hrs.txt

{"created_at":"Fri Apr 05 17:42:16 +0000 2019","id":1114221731600830465,"id_str":"1114221731600830465","text":"RT @Fahrenthold: The IRS commissioner also owns units in a Trump bldg in Hawaii. https:\/\/t.co\/83Zi28DAH8","source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":23220889,"id_str":"23220889","name":"Helen Rosenthal","screen_name":"HelenRosenthal","location":"New York City","url":"http:\/\/HelenRosenthal.com","description":"Councilmember for NYC's Council District 6. Upper West Side Community Leader, Mother, Wife, and Friend.","translator_type":"none","protected":false,"verified":true,"followers_count":11806,"friends_count":2822,"listed_count":279,"favourites_count":14652,"statuses_count":15168,"created_at":"Sat Mar 07 18:48:30 +0000 2009","utc_offset":null,"time_zone":null,"geo_enabled":true,"lang":"en","contributors_enabled":false,"is_translator":false,"profile_background_color":"203187","profile_background_image_url":"http:\/\/abs.twimg.com\/images\/themes\/theme16\/bg.gif","profile_background_image_url_https":"https:\/\/abs.twimg.com\/images\/themes\/theme16\/bg.gif","profile_background_tile":false,"profile_link_color":"0084B4","profile_sidebar_border_color":"FFFFFF","profile_sidebar_fill_color":"E3E2DE","profile_text_color":"634047","profile_use_background_image":true,"profile_image_url":"http:\/\/pbs.twimg.com\/profile_images\/528293522424594433\/Tz2ytHrp_normal.jpeg","profile_image_url_https":"https:\/\/pbs.twimg.com\/profile_images\/528293522424594433\/Tz2ytHrp_normal.jpeg","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banners\/23220889\/1458144124","default_profile":false,"default_profile_image":false,"following":null,"follow_request_sent":null,"notifications":null},"geo":null,"coordinates":null,"place":null,"contributors":null,"retweeted_status":{"created_at":"Thu Apr 04 21:50:54 +0000 2019","id":1113921913997406208,"id_str":"1113921913997406208","text":"The IRS commissioner also owns units in a Trump bldg in Hawaii. https:\/\/t.co\/83Zi28DAH8","display_text_range":[0,63],"source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":61734492,"id_str":"61734492","name":"David Fahrenthold","screen_name":"Fahrenthold","location":"Washington, DC","url":"https:\/\/www.washingtonpost.com\/politics\/7-key-questions-about-what-president-trumps-company-faces-in","description":"Washington Post reporter, covering President Trump's businesses and conflicts of interest. MSNBC contributor. Tips? Fahrenthold@washpost.com","translator_type":"none","protected":false,"verified":true,"followers_count":634034,"friends_count":4499,"listed_count":8156,"favourites_count":9808,"statuses_count":42220,"created_at":"Fri Jul 31 09:29:37 +0000 2009","utc_offset":null,"time_zone":null,"geo_enabled":true,"lang":"en","contributors_enabled":false,"is_translator":false,"profile_background_color":"C0DEED","profile_background_image_url":"http:\/\/abs.twimg.com\/images\/themes\/theme1\/bg.png","profile_background_image_url_https":"https:\/\/abs.twimg.com\/images\/themes\/theme1\/bg.png","profile_background_tile":false,"profile_link_color":"1DA1F2","profile_sidebar_border_color":"C0DEED","profile_sidebar_fill_color":"DDEEF6","profile_text_color":"333333","profile_use_background_image":true,"profile_image_url":"http:\/\/pbs.twimg.com\/profile_images\/1021824606087331841\/_gPQBRax_normal.jpg","profile_image_url_https":"https:\/\/pbs.twimg.com\/profile_images\/1021824606087331841\/_gPQBRax_normal.jpg","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banners\/61734492\/1478793649","default_profile":true,"default_profile_image":false,"following":null,"follow_request_sent":null,"notifications":null},"geo":null,"coordinates":null,"place":null,"contributors":null,"quoted_status_id":1113919903977938945,"quoted_status_id_str":"1113919903977938945","quoted_status":{"created_at":"Thu Apr 04 21:42:55 +0000

We want to capture this data into a file that we will use later for the analysis. We can do so by piping the output to a file using the following command:

```
python twitter_live_v1_keyword.py > twitter_live_v1_keyword.txt
```

I run the program and it gets 'Read Timed Out' after 30mins approx. (The pricing for the premium APIs ranges from **$149**/month to**$2,499**/month, based on the level of access needed.

```
python twitter_live_v1_udemy.py > twitter_live_v1_udemy.txt
python twitter_live_v1_coursera.py > twitter_live_v1_coursera.txt
python twitter_live_v1_BJP.py > twitter_live_v1_BJP.txt
python twitter_live_v1_congress.py > twitter_live_v1_congress.txt
```

The data that we stored `twitter_live_v1_keyword.txt` is in JSON format. This format makes it easy to humans to read the data, and for machines to parse it. Below is an example for one tweet in JSON format. You can see that the tweet contains additional information in addition to the main text which in this example: `"Prime Minister roasted ABP on Rafale and their compulsion not to ask sharp questions to Rahul Gandhi, the liar. He also asked why the channel blacked out Finance Minister\u2019s press conference on Gandhi\u2019s corruption in an online mag. Both interviewers ended up fumbling. #FreePress?"`

1356584215","default_profile":false,"default_profile_image":false,"following":null,"follow_request_sent":null,"notifications":null},"geo":null,"coordinates":null,"place":null,"contributors":null,"is_quote_status":false,"extended_tweet":{"full_text":"Prime Minister roasted ABP on Rafale and their compulsion not to ask sharp questions to Rahul Gandhi, the liar. He also asked why the channel blacked out Finance Minister\u2019s press conference on Gandhi\u2019s corruption in an online mag. Both interviewers ended up fumbling. #FreePress?","display_text_range":[0,279],"entities":{"hashtags":[{"text":"FreePress","indices":[268,278]}],"urls":[],"user_mentions":[],"symbols":[]}},"quote_count":12,"reply_count":33,"retweet_count":281,"favorite_count":609,"entities":{"hashtags":[],"urls":[{"url":"https:\/\/t.co\/QzdE9PHTD4","expanded_url":"https:\/\/twitter.com\/i\/web\/status\/1114210760803213313","display_url":"twitter.com\/i\/

web\/status\/1\u2026","indices":[116,139]}],"user_mentions":[],"symbols":[]},"favorited":false,"retweeted":false,"filter_level":"low","lang":"en"},"is_quote_status":false,"quote_count":0,"reply_count":0,"retweet_count":0,"favorite_count":0,"entities":{"hashtags":[],"urls":[],"user_mentions":[{"screen_name":"amitmalviya","name":"Chowkidar Amit Malviya","id":95588504,"id_str":"95588504","indices":[3,15]}],"symbols":[]},"favorited":false,"retweeted":false,"filter_level":"low","lang":"en","timestamp_ms":"1554486142038"}

Next we will read the data in into an array that we call `tweets` .

Next, we will structure the tweets data into a pandas DataFrame to simplify the data manipulation. We will start by creating an empty DataFrame called `tweets` using the following command.

We need to perform sanity check to verify relevant tweets are captured and in case of any deviations take appropriate steps based on EDA.

The extracted tweets for keywords : 'INC', 'Indian National Congress','Rahul Gandhi'

```
['RT 11',
 'The exit of the Iron Man',
 'RT Utterly shameful that this person has cooked up a story just to demonize hindus and on you woman for using',
 'एक',
 'that Not only but also all religions including',
 'said I Love Mr Narendra he Was He Said The anyone Drunked he said the Truth',
 'RT Is Narendra Modi the best Prime Minister of India till',
 'RT It is not that is contesting in It is the Muslim we are in 1947',
 'RT The area around Sri Harmandir Sahib was chaotic crowded before the went in for a complete The results',
 'RT तप ऊपर',
 'RT worry about This time I will not leave will kill them even if Congess party removes me or',
 'RT 1985 Narendra Modi Holding Rifle While On a Trip to Mount Photo India Today Magazine',
 'RT आप समय एक तरह',
 'RT BJP और आम EVM पर',
 'RT जनरल',
 'RT 650 BJP और न',
 'RT आप समय एक तरह',
 'RT BJP व BJP',
```

(ii) Amazon Review Data :

400K text reviews and ratings for testing

2 Methodology - Twitter Sentiment Analysis

2.1 Exploratory Analysis

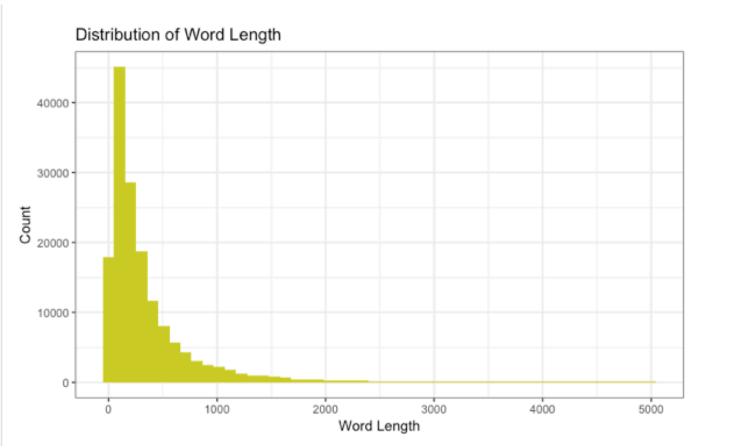
Data : Twitter Sentiment Analysis

Any predictive modeling requires that we look at the data before we start modeling. However, in text mining terms looking at data refers to so much more than just looking. Looking at text refers to exploring the text, cleaning the text data as well as visualizing the text data through graphs and plots. To start this process we will first clean the data by removing irrelevant , meaningless texts or characters which do not add valuable information to text data. than. Further, We can visualize that in a glance by looking at the frequency or term factors of the text.

Preprocessing is an important task and critical step in Text mining. In the area of Text Mining, data preprocessing used for extracting interesting and non-trivial and knowledge from unstructured text data. Information Retrieval (IR) is essentially a matter of deciding which documents in a collection should be retrieved to satisfy a user's need for information. Before the information retrieval from the documents, the data preprocessing techniques are applied on the target data set to reduce the size of the data set which will increase the effectiveness of IR System.

Distribution of Word Length :

In figure below we have plotted a histogram showing the comment word length distribution. As visible in histogram , distribution is right skewed with maximum comments with word length case to a few hundreds.



Distribution of Word Length(See R Code in Appendix)

Most Frequent Words

We generate list of the most frequently occurring words in the comments (excluding stopwords).

```
# A tibble: 10 x 2
  word          n
  <fct>       <int>
 1 article     55907
 2 page        46189
 3 wikipedia   36640
 4 talk        32566
 5 edit        18237
 6 people      17835
 7 articles    16123
```

8	time	15841
9	information	12147
10	deletion	11375

Most Frequent Words (See R Code in Appendix)

Tokenisation of the Comments

The comments are broken up into words.The first 10 rows of comments broken up into words are shown below.

RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help 46% Tue 1 May 6:38 PM

RStudio: Notebook Output

toxic <int>	severe_toxic <int>	obscene <int>	threat <int>	insult <int>	identity_hate <int>	word <chr>	n <int>
0	0	0	0	0	0	the	473363
0	0	0	0	0	0	to	279805
0	0	0	0	0	0	of	211851
0	0	0	0	0	0	and	206454
0	0	0	0	0	0	a	193044
0	0	0	0	0	0	i	180537
0	0	0	0	0	0	you	167678
0	0	0	0	0	0	is	162500
0	0	0	0	0	0	that	145778
0	0	0	0	0	0	in	137817

10 rows

Tokenisation of the Comments (See R Code in Appendix)

Unique Categories of Text

The combinations of `toxic,severe toxic,obscene,threat,insult and identity hate` will create unique categories. We will display those categories here. There are 41 unique categories generated.

RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help 43% Tue 1 May 6:50 PM

RStudio: Notebook Output

toxic <int>	severe_toxic <int>	obscene <int>	threat <int>	insult <int>	identity_hate <int>	total <int>
0	0	0	0	0	0	9882824
0	0	0	0	0	1	4843
0	0	0	0	1	0	17491
0	0	0	0	1	1	1801
0	0	0	1	0	0	765
0	0	0	1	1	0	934
0	0	1	0	0	0	30049
0	0	1	0	0	1	136
0	0	1	0	1	0	13349
0	0	1	0	1	1	1279
0	0	1	1	0	0	204
0	0	1	1	1	0	319
1	0	0	0	0	0	334152
1	0	0	0	0	1	3956
1	0	0	0	1	0	53390
1	0	0	0	1	1	5801
1	0	0	1	0	0	2067

Unique Categories of Toxicity (See R Code Appendix)

TF-IDF

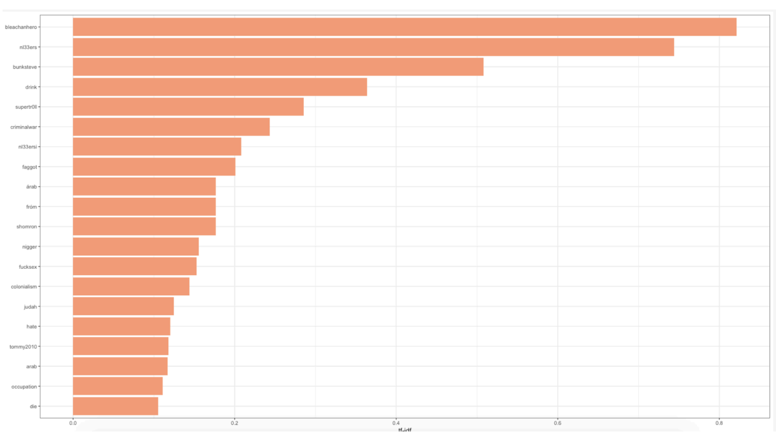
We wish to find out the important words in this `Toxic Comments`. Example for a patient , the most important word is **medicine**. Example for a cook, important words would be related to **food**.
We would explore this using a fascinating concept known as **Term Frequency - Inverse Document Frequency**.

A **document** in this case is the set of lines associated with a unique category determined by the various elements such as `toxic,severe toxic,obscene,threat,insult and identity hate`.
TF-IDF computes a weight which represents the importance of a term inside a document.

It does this by comparing the frequency of usage inside an individual document as opposed to the entire data set (a collection of documents).
The importance increases proportionally to the number of times a word appears in the individual document itself--this is called Term Frequency. However, if multiple documents contain the same word many times then you run into a problem. That's why TF-IDF also offsets this value by the frequency of the term in the entire document set, a value called Inverse Document Frequency.
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).
Value = TF * IDF

Twenty Most Important words

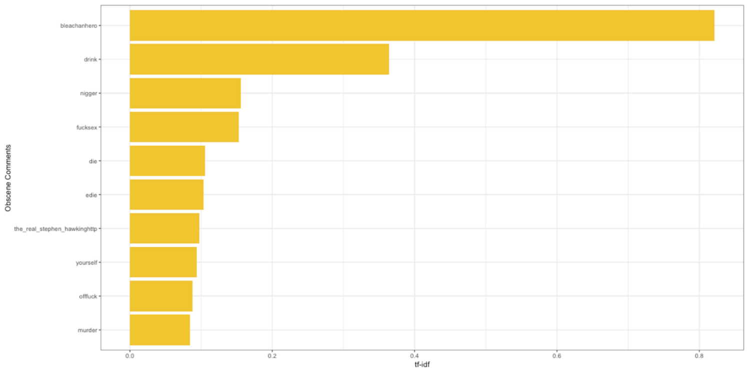
Here using **TF-IDF** , we investigate the **Twenty Most Important words**

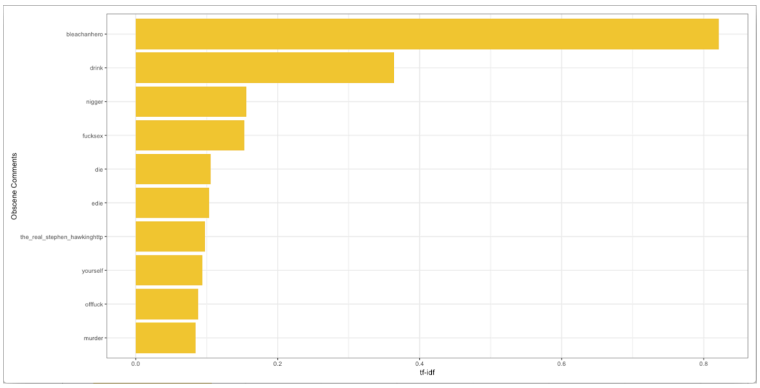


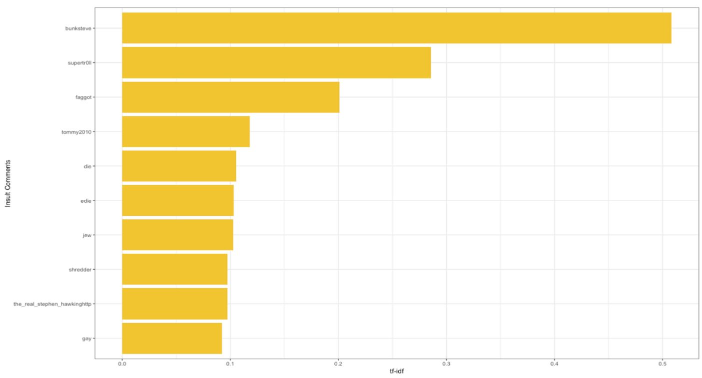
TF-IDF , Important Words   (See R Code in Appendix)

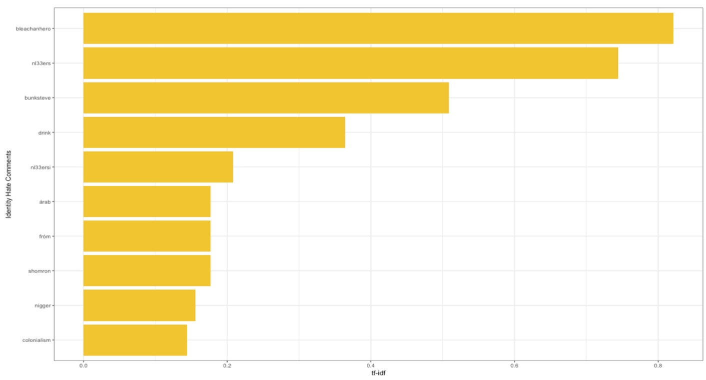
We plot the TF-IDF for the Toxic Comments for each of the 6 categories

- `toxic`
- `severe_toxic`
- `obscene`
- `threat`
- `insult`
- `identity_hate`

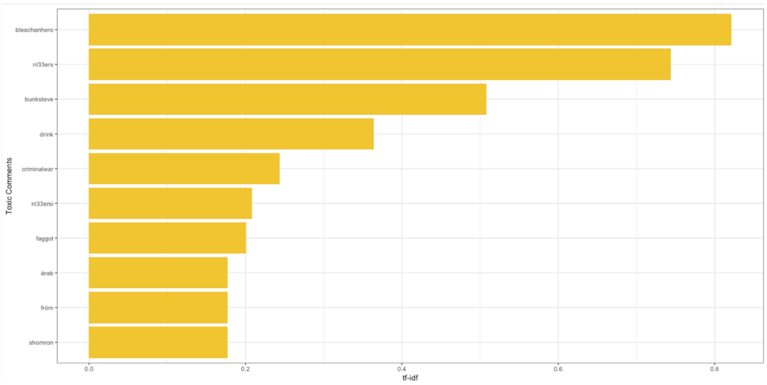


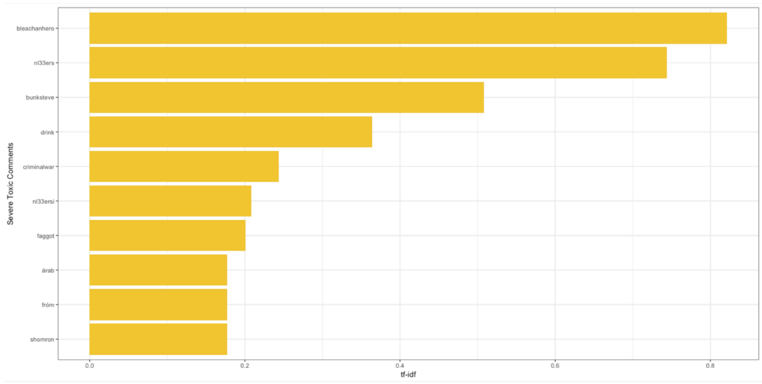






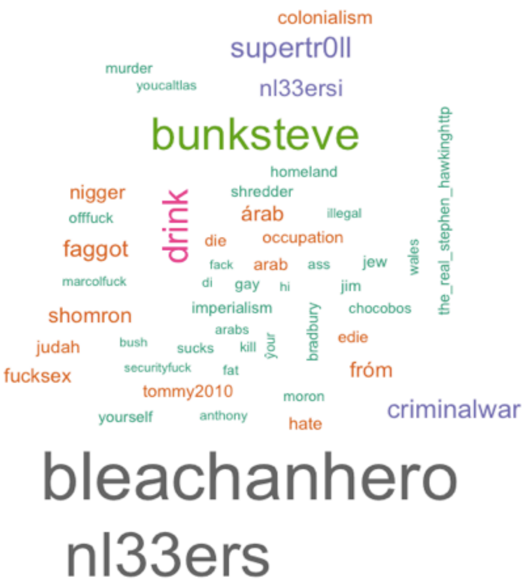
TF-IDF plot, Toxicity Category wise (See R Code in Appendix)





<u>Word Cloud for the Most Important Words</u>

We show the **Fifty** most important words. This Word Cloud is based on the **TF- IDF** scores. Higher the score, bigger is the size of the text.



Word Cloud (See R Code in Appendix)

2.2 Pre Processing

We incorporate text pre-processing techniques

a) Punctuation Marks - Remove punctuation marks from text. Like ? ! , ; [] () <>, . Also need to be careful for change in context of text getting changed due to removal, for example mr. john to mr

John (mr stands for mister in 1st instance but could be misinterpreted as medical representative in 2nd)

b) Numbers - Remove numbers from the text content available, for example

3/12/91
Mar 13 1991
55 B.C
B-52
100.2.86.144

c) Case Folding - The whole point of lowercasing terms is to make them *more* likely to match, this job is done by case folding rather than by lowercasing. *Case folding* is the act of converting words into a (usually lowercase) form that does not necessarily result in the correct spelling, but does allow case-insensitive comparisons.

For instance, the letter ß, which is already lowercase, is *folded* to ss. Similarly, the lowercase ς is folded to σ, to make σ, ς, and Σ comparable, no matter where the letter appears in a word.

d) Stop Words - Many words in documents recur very frequently but are essentially meaningless as they are used to join words together in a sentence. It is commonly understood that stop words do not contribute to the context or content of textual documents. Due to their high frequency of occurrence, their presence in text mining presents an obstacle in understanding the content of the documents.

Stop words are very frequently used common words like ‘and’, ‘are’, ‘this’ etc. They are not useful in classification of documents. So they must be removed. This process also reduces the text data and improves the system performance.

e) White Spaces - Removes unnecessary extra space characters from text.

f) Stemming - Stemming is the process of conflating the variant forms of a word into a common representation, the stem. For example, the words: “presentation”, “presented”, “presenting” could all be reduced to a common representation “present”.

Both train and test data need to be pre-processed.

```
# Delete the leading spaces
library(stringr)
train$comment_text = str_trim(train$comment_text)

# Convert comment into corpus
library(tm)
traincorpus = Corpus(VectorSource(train$comment_text))

# Case Folding
traincorpus = tm_map(traincorpus, tolower)
# Remove Stop Words
traincorpus = tm_map(traincorpus,removeWords,stopwords('english'))
# Remove Punctuation marks
traincorpus = tm_map(traincorpus,removePunctuation)
# Remove Numbers
traincorpus = tm_map(traincorpus,removeNumbers)
# Remove unnecessary spaces
traincorpus = tm_map(traincorpus,stripWhitespace)
# Stemming
traincorpus = tm_map(traincorpus, stemDocument)
```

Now, We create the DTM for both Test and Train dataset. Sparse the dataset to remove words with less than 1% of occurrences to improve the efficiency of the model and decrease running time. Also, We need only those columns information from train dataset which are present in test dataset and only those columns in test dataset which can gather any meaningful insight on toxicity from train data set. Thus, we select common columns from both datasets

Project1_V2.Rmd* | test_dataset | plot_trainWords | total_words | trainWords

Filter

	dont	ever	get	guy	like	man	next	right	rule	thing	tim
1	1	1	2	1	1	1	1	1	2	1	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	

Project1_V2.Rmd* | test_dataset | train_dataset

Filter

	edit	just	made	new	now	page	pleas	remov
1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0
3	2	1	0	0	0	1	0	1
4	0	0	0	0	0	0	1	0
5	0	0	0	0	0	1	0	0

screenshot shoeing DTM for common columns (See R Code in Appendix)

2.3 Modelling

In our early stages of analysis during pre-processing we have come to understand the words in comments that attribute to different toxicity levels in the train dataset. Based on the words in the every document probability can be assigned to the type of toxic category. We Fit Predictive Models over Different Tuning Parameters. Tuning Parameters can be derived for each toxic category.

We will be using XGBoost algorithm to calculate the various targets and predict the probabilities for each type of toxicity.

```
dataset2 = dataset
dataset2$toxic = train$toxic
dataset2$toxic = as.factor(dataset2$toxic)
levels(dataset2$toxic) = make.names(unique(dataset2$toxic))

formula = toxic ~ .
```

```
fitControl <- trainControl(method="none",classProbs=TRUE,
summaryFunction=twoClassSummary)

xgbGrid <- expand.grid(nrounds = 500, max_depth = 3, eta = .05,   gamma = 0,

                        colsample_bytree = .8, min_child_weight = 1,
                        subsample = 1)

set.seed(13)

ToxicXGB = train(formula, data = dataset2,
        method = "xgbTree",trControl = fitControl,
        tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

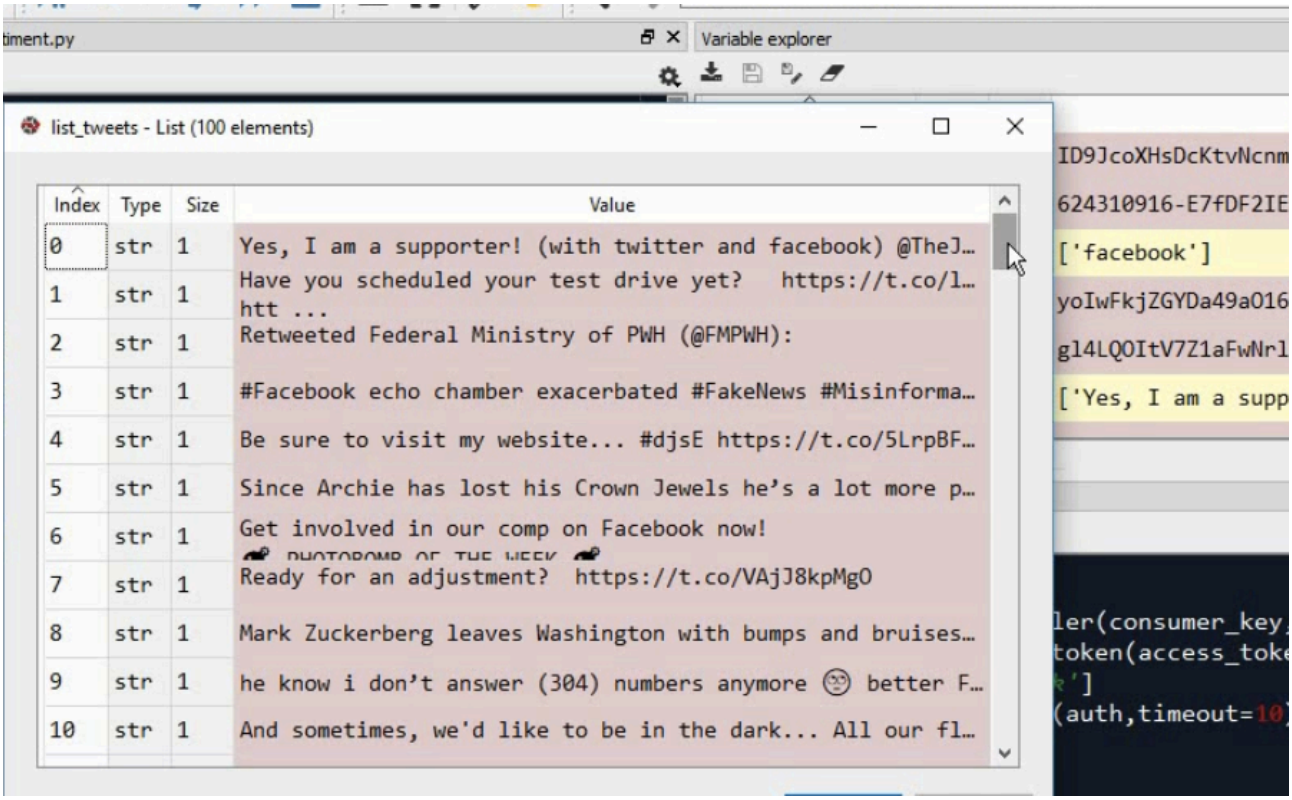
predictionsToxic = predict(ToxicXGB,datasetTest,type = 'prob')
```

3 Methodology - Twitter 'trump' Sentiment

I am fetching tweets in realtime from twitter and we will use our classifier to perform sentiment analysis on those tweets.

I have fetched top hundred recent tweets about 'trump' from twitter.

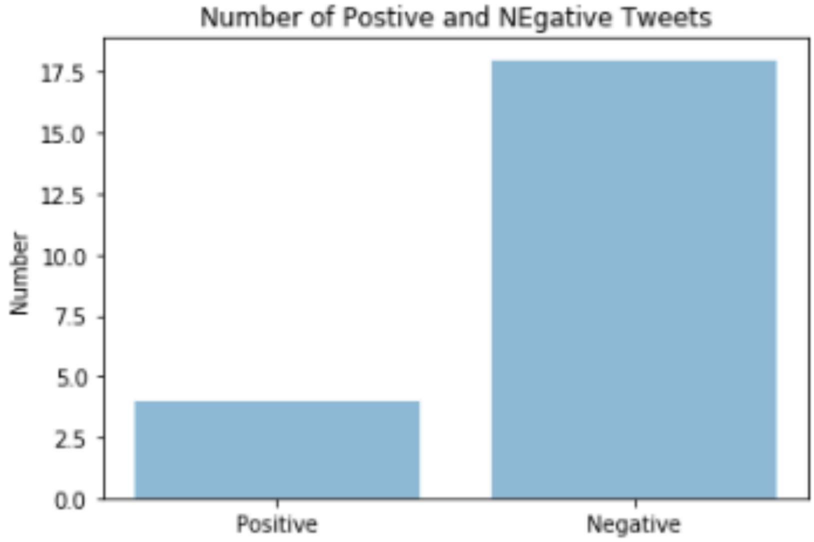
The loaded data will be as below :



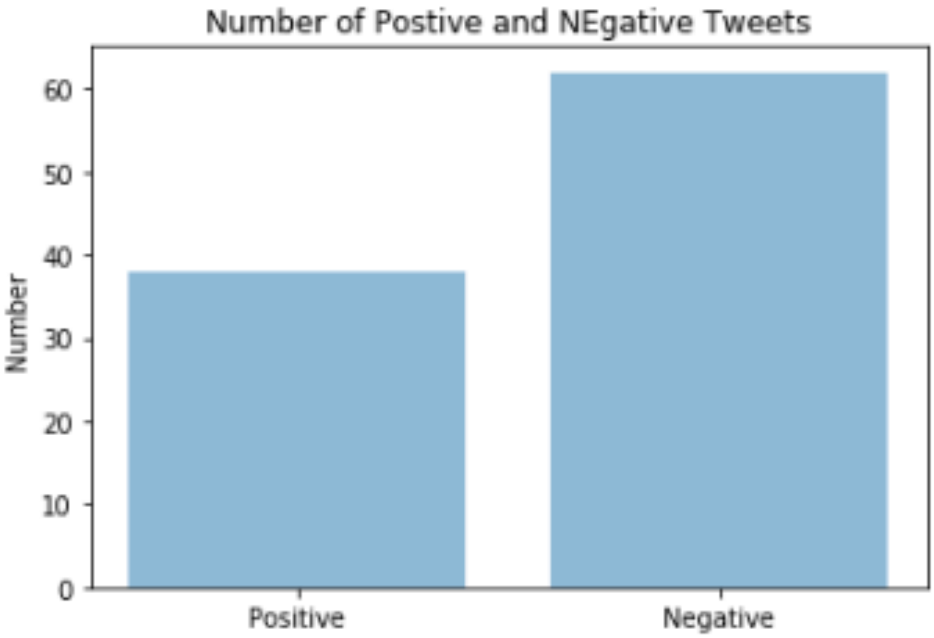
We need to pre-process all the different types of tweets as it contains a lot of impurities like links, different symbols, punctuation marks and so on. We have to pretty much delete all of unrequited data and generate fresh text from it.

Visualizing the sentiment:

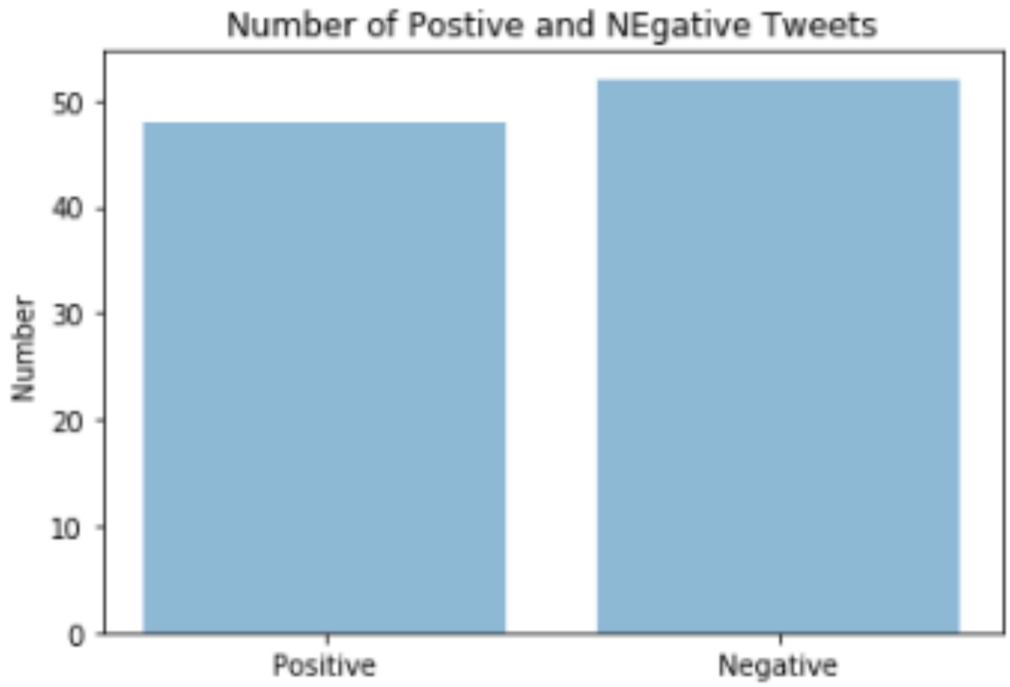
‘trump’



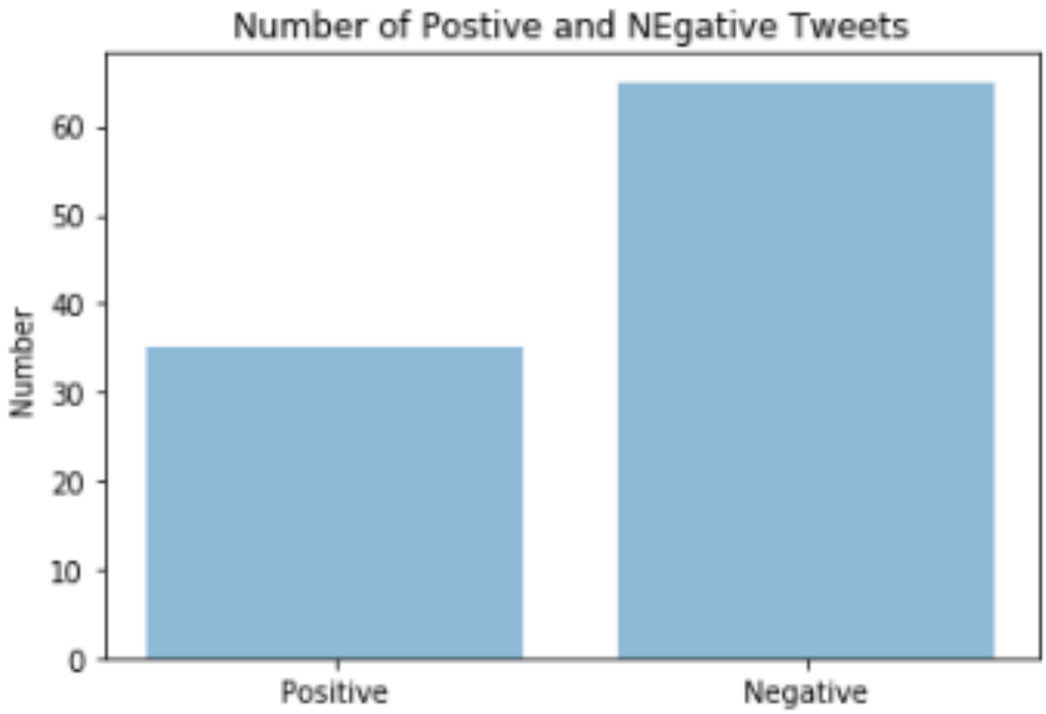
‘modi’



‘rahulgandhi’


Number of Postive and NEgative Tweets
Number
0
10
20
30
40
50
Positive
Negative

'pappu'


Number of Postive and NEgative Tweets
Number
0
10
20
30
40
50
60
Positive
Negative

4 Methodology - Perceptron - Mobile Like Unlike Classification

See Appendix D Perceptron - Mobile Like Unlike Classification

Appendix A - Twitter Data Preparation Python Code

```
from tweepy import Stream
#Variables that contains the user credentials to access Twitter API
access_token = "ENTER YOUR ACCESS TOKEN"
access_token_secret = "ENTER YOUR ACCESS TOKEN SECRET"
consumer_key = "ENTER YOUR API KEY"
consumer_secret = "ENTER YOUR API SECRET"
#This is a basic listener that just prints received tweets to stdout.
class StdOutListener(StreamListener):
    def on_data(self, data):
        print data
return True
    def on_error(self, status):
        print status
    if __name__ == '__main__':

            #This handles Twitter authetification and the connection to Twitter Streaming API
            l = StdOutListener()
            auth = OAuthHandler(consumer_key, consumer_secret)
            auth.set_access_token(access_token, access_token_secret)
            stream = Stream(auth, l)
            #This line filter Twitter Streams to capture data by the keywords: 'INC', 'Indian National
            #Congress','Rahul Gandhi'
            stream.filter(track=[''INC', 'Indian National Congress','Rahul Gandhi'])

import json
import pandas as pd
import matplotlib.pyplot as plt

tweets_data_path = '/Users/abhishek/Desktop/Infosys_Hackathon_2019/
twitter_live_v1_BJP_5apr2300hrs.txt'
tweets_data = []
tweets_file = open(tweets_data_path, "r")
for line in tweets_file:
   try:
       tweet = json.loads(line)
       tweets_data.append(tweet)
   except:
       continue

tweets = pd.DataFrame()
tweets['text'] = map(lambda tweet: tweet['text'], tweets_data)
tweets['lang'] = map(lambda tweet: tweet['lang'], tweets_data)
```

```
tweets['country'] = map(lambda tweet: tweet['place']['country'] if tweet['place'] != None else None,
tweets_data)
tweets_by_lang = tweets['lang'].value_counts()

list1 = []
for data in tweets_data:
    if (data['text']).isalnum:
        list1.append(data['text'])

extracted_tweets = []

for data in list1:
    list3 = []
    for text in list(data.split(' ')):
        if text.isalnum():
            list3.append(text)
    extracted_tweets.append(' '.join(list3))
```

Appendix B - Twitter ‘trump’ Sentiment

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


# Twitter Sentiment Analysis using NLP

# Install tweepy - pip install tweepy

# Importing the libraries
import tweepy
import re
import pickle
import matplotlib.pyplot as plt
from tweepy import OAuthHandler

# Please change with your own consumer key, consumer secret, access token and access secret
# Initializing the keys
access_token = "1065497629767991296-bIp1dADwScXufcJn3Px1hJoksR84T3"
access_secret = "1BlMiT1DYtO6aRoMbG7H93ZmrS53YCREUSSp68IKdaotc"
consumer_key = "kGNm4K8DhGZBqzKmWEUsozgZQ"
consumer_secret = "tO4pWXf0X4rlKfvzvayx0xBpaaUzxNyJTluJyuTJa1Cu9FkvSW"

# Initializing the tokens
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
args = ['trump'];
api = tweepy.API(auth,timeout=10)

# Fetching the tweets
list_tweets = []

query = args[0]
if len(args) == 1:
    for status in tweepy.Cursor(api.search,q=query+" -
filter:retweets",lang='en',result_type='recent',geocode="22.1568,89.4332,500km").items(100):
```

```
        list_tweets.append(status.text)

# Loading the vectorizer and classfier
with open('classifier.pickle','rb') as f:
    classifier = pickle.load(f)

with open('tfidfmodel.pickle','rb') as f:
    tfidf = pickle.load(f)

total_pos = 0
total_neg = 0

# Preprocessing the tweets and predicting sentiment
for tweet in list_tweets:
    tweet = re.sub(r"^https://t.co/[a-zA-Z0-9]*\s", " ", tweet)
    tweet = re.sub(r"\s+https://t.co/[a-zA-Z0-9]*\s", " ", tweet)
    tweet = re.sub(r"\s+https://t.co/[a-zA-Z0-9]*$", " ", tweet)
    tweet = tweet.lower()
    tweet = re.sub(r"that's","that is",tweet)
    tweet = re.sub(r"there's","there is",tweet)
    tweet = re.sub(r"what's","what is",tweet)
    tweet = re.sub(r"where's","where is",tweet)
    tweet = re.sub(r"it's","it is",tweet)
    tweet = re.sub(r"who's","who is",tweet)
    tweet = re.sub(r"i'm","i am",tweet)
    tweet = re.sub(r"she's","she is",tweet)
    tweet = re.sub(r"he's","he is",tweet)
    tweet = re.sub(r"they're","they are",tweet)
    tweet = re.sub(r"who're","who are",tweet)
    tweet = re.sub(r"ain't","am not",tweet)
    tweet = re.sub(r"wouldn't","would not",tweet)
    tweet = re.sub(r"shouldn't","should not",tweet)
    tweet = re.sub(r"can't","can not",tweet)
    tweet = re.sub(r"couldn't","could not",tweet)
    tweet = re.sub(r"won't","will not",tweet)
    tweet = re.sub(r"\W"," ",tweet)
    tweet = re.sub(r"\d"," ",tweet)
    tweet = re.sub(r"\s+[a-z]\s+"," ",tweet)
    tweet = re.sub(r"\s+[a-z]$"," ",tweet)
    tweet = re.sub(r"^[a-z]\s+"," ",tweet)
    tweet = re.sub(r"\s+"," ",tweet)
    sent = classifier.predict(tfidf.transform([tweet]).toarray())
    print(tweet,':',sent)
    if sent[0] == 1:
        total_pos += 1
    else:
        total_neg += 1



# In[3]:



# Visualizing the results
import matplotlib.pyplot as plt
import numpy as np
objects = ['Positive','Negative']
y_pos = np.arange(len(objects))

plt.bar(y_pos,[total_pos,total_neg],alpha=0.5)
```

```
plt.xticks(y_pos,objects)
plt.ylabel('Number')
plt.title('Number of Postive and NEgative Tweets')

plt.show()


# In[ ]:
```

Appendix C - Twitter Sentiment Analysis

```
#Figure 2.1 : Distribution of Word Length

train$len = str_count(train$comment_text)
test$len = str_count(test$comment_text)

train %>%
  ggplot(aes(x = len)) +
  geom_histogram(fill= 'yellow3',bins = 50) +
  labs(x= 'Word Length',y = 'Count', title = 'Distribution of Word Length') +
  theme_bw()

#Table 2.1 : Most Frequent Words

  train %>%
    unnest_tokens(word, comment_text) %>%
    filter(!word %in% stop_words$word) %>%
    count(word,sort = TRUE) %>%
    ungroup() %>%
    head(10)


#Table 2.2 : Tokenisation of the Comments

trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word) %>%
  ungroup()

head(trainWords,10)


#Table 2.3 : Unique Categories of Toxicity

total_words <- trainWords %>%
  group_by(toxic,severe_toxic,obscene,threat,insult,identity_hate) %>%
  summarize(total = sum(n))

total_words
```

```
#Figure 2.2 : TF-IDF , Important Words

Category =1:41

total_words$Category = Category

trainWords <- left_join(trainWords, total_words)

trainWords <- trainWords %>%
  bind_tf_idf(word, Category, n)


plot_trainWords <- trainWords %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

plot_trainWords %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'orange') +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()

#Figure 2.3 : TF-IDF plot, Toxicity Category wise

# For 'toxic' comment
plot_trainWords %>%
  filter(toxic == 1 ) %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = fillColor2) +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()


#For 'Severe Toxic' Comment
plot_trainWords %>%
  filter(severe_toxic == 1 ) %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = fillColor2) +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()


#Figure 2.4 : Word Cloud

plot_trainWords %>%
  with(wordcloud(word, tf_idf, max.words = 50,colors=brewer.pal(8, "Dark2")))

#Figure 2.5 : screenshot shoeing DTM for common columns

colnamesSame = intersect(colnames(dataset),colnames(datasetTest))
dataset = dataset[ , (colnames(dataset) %in% colnamesSame)]
```

```
datasetTest = datasetTest[ , (colnames(datasetTest) %in% colnamesSame)]


Complete R Code : (*.Rmd)

#Load Library, setwd, import data files
```{r,message=FALSE,warning=FALSE}

library(tidyverse)
library(tidytext)
install.packages("tidytext",   lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
library(DT)
library(stringr)
library('wordcloud')
install.packages("igraph", lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/library")
library(igraph)
install.packages("ggraph",   lib="/Library/Frameworks/R.framework/Versions/3.4/Resources/
library")
library(ggraph)
library(tm)
library(SnowballC)
library(caret)

rm(list=ls())

# setwd('/Users/abhishek/Desktop/Edwisor_Data Science Career/Aggregate_Notes/Project_1')

train = read_csv("train.csv")
test = read_csv("test.csv")
submission = read_csv("sample_submission.csv")

```

View the Data
```{r,message=FALSE,warning=FALSE}

head(train)

```

Word Length Distribution
```{r,message=FALSE,warning=FALSE}

train$len = str_count(train$comment_text)
test$len = str_count(test$comment_text)

train %>%
  ggplot(aes(x = len)) +
  geom_histogram(fill= 'yellow2',bins = 50) +
  labs(x= 'Word Length',y = 'Count', title = paste('Distribution of Word Length ')) +
  theme_bw()

```


#Top Ten most Common Words
```{r,message=FALSE,warning=FALSE}
```
```

````
 train %>%
   unnest_tokens(word, comment_text) %>%
   filter(!word %in% stop_words$word) %>%
   count(word,sort = TRUE) %>%
   ungroup() %>%
   # mutate(word = factor(word, levels = rev(unique(word)))) %>%
   head(10)


```


#Tokenisation of the sentences The sentences are broken up into words as shown below.

```{r,message=FALSE,warning=FALSE}

trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word, sort = TRUE) %>%
  ungroup()

head(trainWords,10)
```


#Unique Categories of Text The combinations of `toxic,severe toxic,obscene,threat,insult and # #
#identity hate` will create unique categories. We will display those categories here.

```{r,message=FALSE,warning=FALSE}

trainWords <- train %>%
  unnest_tokens(word, comment_text) %>%
  count(toxic,severe_toxic,obscene,threat,insult,identity_hate,word) %>%
  ungroup()

total_words <- trainWords %>%
  group_by(toxic,severe_toxic,obscene,threat,insult,identity_hate) %>%
  summarize(total = sum(n))

total_words


```


#TF-IDF
## Twenty Most Important words Here using **TF-IDF** , we investigate the **Twenty Most
#Important words**
```{r, message=FALSE, warning=FALSE}

Category =1:41

total_words$Category = Category

trainWords <- left_join(trainWords, total_words)

#Now we are ready to use the bind_tf_idf which computes the tf-idf for each term.
trainWords <- trainWords %>%
  bind_tf_idf(word, Category, n)
````

````
plot_trainWords <- trainWords %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

plot_trainWords %>%
  top_n(20) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'orange') +
  labs(x = NULL, y = "tf-idf") +
  coord_flip() +
  theme_bw()


```



#Various Categories of TF-IDF

##Toxic TF-IDF We plot the TF-IDF for the Toxic Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(toxic == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Toxic Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```



##Severe Toxic TF-IDF
We plot the TF-IDF for the Severe Toxic Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(severe_toxic == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Severe Toxic Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```

##Obscene TF-IDF We plot the TF-IDF for the Obscene Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(obscene == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
````

````
  geom_col(fill = 'yellow') +
  labs(x = 'Obscene Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()

```


##Threat TF-IDF
We plot the TF-IDF for the Threat Comments
```{r,message=FALSE,warning=FALSE}


plot_trainWords %>%
  filter(threat == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Threat Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()


# For 'insult' Comment

plot_trainWords %>%
  filter(insult == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Insult Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()


# For 'identity hate' Comment

plot_trainWords %>%
  filter(identity_hate == 1 ) %>%
  top_n(10) %>%
  ggplot(aes(word, tf_idf)) +
  geom_col(fill = 'yellow') +
  labs(x = 'Identity Hate Comments', y = "tf-idf") +
  coord_flip() +
  theme_bw()
```



#Word Cloud for the Most Important Words We show the **Fifty** most important words. This
#Word Cloud is based on the **TF- IDF** scores. Higher the score, bigger is the size of the text.
```{r, message=FALSE, warning=FALSE}

plot_trainWords %>%
  with(wordcloud(word, tf_idf, max.words = 50,colors=brewer.pal(8, "Dark2")))

```



#Pre-rocessing
```{r,message =FALSE,warning=FALSE}
````

```
# Delete the leading spaces
library(stringr)
train$comment_text = str_trim(train$comment_text)
class(train$comment_text) # Class is 'Charcter'

# Convert comment into corpus
library(tm)
traincorpus = Corpus(VectorSource(train$comment_text))
writeLines(as.character(train$comment_text[10]))

# Case Folding
traincorpus = tm_map(traincorpus, tolower)
# Remove Stop Words
traincorpus = tm_map(traincorpus,removeWords,stopwords('english'))
# Remove Punctuation marks
traincorpus = tm_map(traincorpus,removePunctuation)
# Remove Numbers
traincorpus = tm_map(traincorpus,removeNumbers)
# Remove unnecessary spaces
traincorpus = tm_map(traincorpus,stripWhitespace)
# Stemming
traincorpus = tm_map(traincorpus, stemDocument)

##############################################################################
#####
test$comment_text = str_trim(test$comment_text)
testcorpus = Corpus(VectorSource(test$comment_text))
testcorpus = tm_map(testcorpus, tolower)
testcorpus = tm_map(testcorpus,removeWords,stopwords('english'))
testcorpus = tm_map(testcorpus,removePunctuation)
testcorpus = tm_map(testcorpus,removeNumbers)
testcorpus = tm_map(testcorpus,stripWhitespace)
testcorpus = tm_map(testcorpus, stemDocument)

##############################################################################
#####

dtm = DocumentTermMatrix(traincorpus)
dtm = removeSparseTerms(dtm, 0.99)
train_dataset = as.data.frame(as.matrix(dtm))
train_dataset$toxic = NULL
train_dataset$severe_toxic = NULL
train_dataset$obscene = NULL
train_dataset$threat = NULL
train_dataset$insult = NULL
train_dataset$identity_hate = NULL

##############################################################################
#####

dtm = DocumentTermMatrix(testcorpus)
dtm = removeSparseTerms(dtm, 0.99)
test_dataset = as.data.frame(as.matrix(dtm))

##############################################################################
########

colnamesSame = intersect(colnames(train_dataset),colnames(test_dataset))

train_dataset = train_dataset[ , (colnames(train_dataset) %in% colnamesSame)]
```

````
test_dataset = test_dataset[ , (colnames(test_dataset) %in% colnamesSame)]


################################################################################
########


```

#Modelling using XGBoost
##Toxic Calculation
We calculate the various targets and predict the probablities
```{r,message=FALSE,warning=FALSE}



dataset2 = train_dataset
dataset2$toxic = train$toxic
dataset2$toxic = as.factor(dataset2$toxic)
levels(dataset2$toxic) = make.names(unique(dataset2$toxic))

formula = toxic ~ .

fitControl <- trainControl(method="none",classProbs=TRUE,
summaryFunction=twoClassSummary)

xgbGrid <- expand.grid(nrounds = 500,
                      max_depth = 3,
                      eta = .05,
                      gamma = 0,
                      colsample_bytree = .8,
                      min_child_weight = 1,
                      subsample = 1)


set.seed(13)

ToxicXGB = train(formula, data = dataset2,
                method = "xgbTree",trControl = fitControl,
                tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsToxic = predict(ToxicXGB,test_dataset,type = 'prob')


################################################################################
######################

```


##Severe Toxic Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$severe_toxic = train$severe_toxic
dataset2$severe_toxic = as.factor(dataset2$severe_toxic)
levels(dataset2$severe_toxic) = make.names(unique(dataset2$severe_toxic))

formula = severe_toxic ~ .
````

```
set.seed(13)

ToxicXGB = train(formula, data = dataset2,
               method = "xgbTree",trControl = fitControl,
               tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsSevereToxic = predict(ToxicXGB,test_dataset,type = 'prob')

########################################################################################

```

##Obscene Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$obscene = train$obscene
dataset2$obscene = as.factor(dataset2$obscene)
levels(dataset2$obscene) = make.names(unique(dataset2$obscene))

formula = obscene ~ .

ObsceneXGB = train(formula, data = dataset2,
               method = "xgbTree",trControl = fitControl,
               tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsObscene = predict(ObsceneXGB,test_dataset,type = 'prob')

########################################################################################

```

##Threat Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$threat = train$threat
dataset2$threat = as.factor(dataset2$threat)
levels(dataset2$threat) = make.names(unique(dataset2$threat))

formula = threat ~ .

ThreatXGB = train(formula, data = dataset2,
                method = "xgbTree",trControl = fitControl,
                tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsThreat = predict(ThreatXGB,test_dataset,type = 'prob')

########################################################################################

```


##Insult Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
```

````
dataset2$insult = train$insult
dataset2$insult = as.factor(dataset2$insult)
levels(dataset2$insult) = make.names(unique(dataset2$insult))

formula = insult ~ .

InsultXGB = train(formula, data = dataset2,
                  method = "xgbTree",trControl = fitControl,
                  tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsInsult = predict(InsultXGB,test_dataset,type = 'prob')

##############################################################################
############

```



##Identity Hate Calculation
```{r,message=FALSE,warning=FALSE}

dataset2 = train_dataset
dataset2$identity_hate = train$identity_hate
dataset2$identity_hate = as.factor(dataset2$identity_hate)
levels(dataset2$identity_hate) = make.names(unique(dataset2$identity_hate))

formula = identity_hate ~ .

HateXGB = train(formula, data = dataset2,
                  method = "xgbTree",trControl = fitControl,
                  tuneGrid = xgbGrid,na.action = na.pass,metric="ROC", maximize=FALSE)

predictionsHate = predict(HateXGB,test_dataset,type = 'prob')

##############################################################################
############

```

#Creating the Submissions
```{r,message=FALSE,warning=FALSE}

submission$toxic = predictionsToxic$X1
submission$severe_toxic = predictionsSevereToxic$X1
submission$obscene = predictionsObscene$X1
submission$threat = predictionsThreat$X1
submission$insult = predictionsInsult$X1
submission$identity_hate = predictionsHate$X1

# Write it to file
write.csv(submission, 'ToxicCommentsSubmission.csv', row.names = F)

```
````
````

Appendix D - Perceptron - Mobile Like Unlike Classification

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, ParameterGrid
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error, log_loss
import operator
import json
from IPython import display
import os
import warnings

np.random.seed(0)
warnings.filterwarnings("ignore")
THRESHOLD = 4
import matplotlib
get_ipython().run_line_magic('matplotlib', 'inline')


# Task: To predict whether the user likes the mobile phone or not. <br>
# Assumption: If the average rating of mobile >= threshold, then the user likes it, otherwise not.

# <b>Missing values:</b><br>
# 'Also Known As'(459),'Applications'(421),'Audio Features'(437),'**Bezel-less display**'(266),'Browser'(449),'Build Material'(338),'Co-Processor'(451),'Display Colour'(457),'Mobile High-Definition Link(MHL)'(472),'Music'(447)
# 'Email','**Fingerprint Sensor Position**'(174),'Games'(446),'HDMI'(454),'Heart Rate Monitor'(467),'IRIS Scanner'(467),
# '**Optical Image Stabilisation**'(219),'Other Facilities'(444),'Phone Book'(444),'Physical Aperture'(87),'**Quick Charging**'(122),'Ring Tone'(444),'Ruggedness'(430),SAR Value(315),'SIM 3'(472),'SMS'(470)', 'Screen Protection'(229),'**Screen to Body Ratio** (claimed by the brand)'(428),'Sensor'(242),'Software Based Aperture'(473),
# '**Special Features**'(459),'Standby time'(334),'Stylus'(473),'TalkTime'(259), '**USB Type-C**'(374),'Video Player'(456),
# 'Video Recording Features'(458),'**Waterproof**'(398),'**Wireless Charging**','**USB OTG Support**'(159), 'Video ,'Recording'(113),'Java'(471),'Browser'(448)
#
# <b>Very low variance:</b><br>
# 'Architecture'(most entries are 64-bit),'Audio Jack','GPS','Loudspeaker','Network','Network Support','Other Sensors'(28),'SIM Size', 'VoLTE'
#
#
# <b>Multivalued:</b><br>
# 'Colours','Custom UI','Model'(1),'Other Sensors','Launch Date'
#
# <b>Not important:</b><br>
# 'Bluetooth', 'Settings'(75),'Wi-Fi','Wi-Fi Features'
#
# <b>Doubtful:</b><br>
```

```
# 'Aspect Ratio','Autofocus','Brand','Camera Features','Fingerprint Sensor'(very few entries are missing),
# 'Fingerprint Sensor Position', 'Graphics'(multivalued),'Image resolution'(multivalued),'SIM Size','Sim Slot(s)', 'User Available Storage', 'SIM 1', 'SIM 2','Shooting Modes', 'Touch Screen'(24), 'USB Connectivity'
#
# <b>To check:</b><br>
# 'Display Type','Expandable Memory','FM Radio'
#
# <b>High Correlation with other features</b><br>
# 'SIM Slot(s)' high correlation with SIM1
# 'Weight' has high high correlation with capacity , screen-to-body ratio
# 'Height' - screen size is also there
#
# <b>Given a mobile, we can't directly get these features</b><br>
# 'Rating Count', 'Review Count'
#
# <b>Keeping:</b><br>
# 'Capacity','Flash'(17),'Height'(22),'Internal Memory'(20, require cleaning),'Operating System'(25, require cleaning), 'Pixel Density'(1, clean it),'Processor'(22, clean it), 'RAM'(17, clean), 'Rating','Resolution'(cleaning), 'Screen Resolution','Screen Size', 'Thickness'(22), 'Type','User Replaceable','Weight'(cleaning),'Sim Size'(), 'Other Sensors'(28), 'Screen to Body Ratio (calculated)','Width',
#

# In[2]:


# read data from file
train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")

# check the number of features and data points in train
print("Number of data points in train: %d" % train.shape[0])
print("Number of features in train: %d" % train.shape[1])

# check the number of features and data points in test
print("Number of data points in test: %d" % test.shape[0])
print("Number of features in test: %d" % test.shape[1])


# In[3]:


def data_clean(data):

    # Let's first remove all missing value features
    columns_to_remove = ['Also Known As','Applications','Audio Features','Bezel-less display'
                    'Browser','Build Material','Co-Processor','Browser'
                    'Display Colour','Mobile High-Definition Link(MHL)',
                    'Music', 'Email','Fingerprint Sensor Position',
                    'Games','HDMI','Heart Rate Monitor','IRIS Scanner',
                    'Optical Image Stabilisation','Other Facilities',
                    'Phone Book','Physical Aperture','Quick Charging',
                    'Ring Tone','Ruggedness','SAR Value','SIM 3','SMS',
                    'Screen Protection','Screen to Body Ratio (claimed by the brand)',
                    'Sensor','Software Based Aperture', 'Special Features',
                    'Standby time','Stylus','TalkTime', 'USB Type-C',
                    'Video Player', 'Video Recording Features','Waterproof',
                    'Wireless Charging','USB OTG Support', 'Video Recording','Java']
```

```
    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]

    #Features having very low variance
        columns_to_remove = ['Architecture','Audio Jack','GPS','Loudspeaker','Network','Network
Support','VoLTE']
    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]

    # Multivalued:
        columns_to_remove = ['Architecture','Launch Date','Audio
Jack','GPS','Loudspeaker','Network','Network Support','VoLTE', 'Custom UI']
    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]

    # Not much important
    columns_to_remove = ['Bluetooth', 'Settings','Wi-Fi','Wi-Fi Features']
    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]

    return data


# # Removing features

# In[4]:


train = data_clean(train)
test = data_clean(test)


# removing all those data points in which more than 15 features are missing

# In[5]:


train = train[(train.isnull().sum(axis=1) <= 15)]
# You shouldn't remove data points from test set
#test = test[(test.isnull().sum(axis=1) <= 15)]


# In[6]:


# check the number of features and data points in train
print("Number of data points in train: %d" % train.shape[0])
print("Number of features in train: %d" % train.shape[1])

# check the number of features and data points in test
print("Number of data points in test: %d" % test.shape[0])
print("Number of features in test: %d" % test.shape[1])


# # Filling Missing values

# In[7]:
```

```
def for_integer(test):
    try:
        test = test.strip()
        return int(test.split(' ')[0])
    except IOError:
            pass
    except ValueError:
        pass
    except:
        pass

def for_string(test):
    try:
        test = test.strip()
        return (test.split(' ')[0])
    except IOError:
        pass
    except ValueError:
        pass
    except:
        pass

def for_float(test):
    try:
        test = test.strip()
        return float(test.split(' ')[0])
    except IOError:
        pass
    except ValueError:
        pass
    except:
        pass
def find_freq(test):
    try:
        test = test.strip()
        test = test.split(' ')
        if test[2][0] == '(':
            return float(test[2][1:])
        return float(test[2])
    except IOError:
        pass
    except ValueError:
        pass
    except:
        pass


def for_Internal_Memory(test):
    try:
        test = test.strip()
        test = test.split(' ')
        if test[1] == 'GB':
            return int(test[0])
        if test[1] == 'MB':
#            print("here")
            return (int(test[0]) * 0.001)
    except IOError:
            pass
    except ValueError:
        pass
```

```
    except:
        pass

def find_freq(test):
    try:
        test = test.strip()
        test = test.split(' ')
        if test[2][0] == '(':
            return float(test[2][1:])
        return float(test[2])
    except IOError:
        pass
    except ValueError:
        pass
    except:
        pass


# In[8]:


def data_clean_2(x):
    data = x.copy()

    data['Capacity'] = data['Capacity'].apply(for_integer)

    data['Height'] = data['Height'].apply(for_float)
    data['Height'] = data['Height'].fillna(data['Height'].mean())

    data['Internal Memory'] = data['Internal Memory'].apply(for_Internal_Memory)

    data['Pixel Density'] = data['Pixel Density'].apply(for_integer)

    data['Internal Memory'] = data['Internal Memory'].fillna(data['Internal Memory'].median())
    data['Internal Memory'] = data['Internal Memory'].astype(int)

    data['RAM'] = data['RAM'].apply(for_integer)
    data['RAM'] = data['RAM'].fillna(data['RAM'].median())
    data['RAM'] = data['RAM'].astype(int)

    data['Resolution'] = data['Resolution'].apply(for_integer)
    data['Resolution'] = data['Resolution'].fillna(data['Resolution'].median())
    data['Resolution'] = data['Resolution'].astype(int)

    data['Screen Size'] = data['Screen Size'].apply(for_float)

    data['Thickness'] = data['Thickness'].apply(for_float)
    data['Thickness'] = data['Thickness'].fillna(data['Thickness'].mean())
    data['Thickness'] = data['Thickness'].round(2)

    data['Type'] = data['Type'].fillna('Li-Polymer')

    data['Screen to Body Ratio (calculated)'] = data['Screen to Body Ratio (calculated)'].apply(for_float)
    data['Screen to Body Ratio (calculated)'] = data['Screen to Body Ratio (calculated)'].fillna(data['Screen to Body Ratio (calculated)'].mean())
    data['Screen to Body Ratio (calculated)'] = data['Screen to Body Ratio (calculated)'].round(2)

    data['Width'] = data['Width'].apply(for_float)
    data['Width'] = data['Width'].fillna(data['Width'].mean())
```

```
    data['Width'] = data['Width'].round(2)

    data['Flash'][data['Flash'].isna() == True] = "Other"

    data['User Replaceable'][data['User Replaceable'].isna() == True] = "Other"

    data['Num_cores'] = data['Processor'].apply(for_string)
    data['Num_cores'][data['Num_cores'].isna() == True] = "Other"


    data['Processor_frequency'] = data['Processor'].apply(find_freq)
    #because there is one entry with 208MHz values, to convert it to GHz
    data['Processor_frequency'][data['Processor_frequency'] > 200] = 0.208
    data['Processor_frequency'] =
data['Processor_frequency'].fillna(data['Processor_frequency'].mean())
    data['Processor_frequency'] = data['Processor_frequency'].round(2)

    data['Camera Features'][data['Camera Features'].isna() == True] = "Other"

    #simplifyig Operating System to os_name for simplicity
    data['os_name'] = data['Operating System'].apply(for_string)
    data['os_name'][data['os_name'].isna() == True] = "Other"

    data['Sim1'] = data['SIM 1'].apply(for_string)

    data['SIM Size'][data['SIM Size'].isna() == True] = "Other"

    data['Image Resolution'][data['Image Resolution'].isna() == True] = "Other"

    data['Fingerprint Sensor'][data['Fingerprint Sensor'].isna() == True] = "Other"

    data['Expandable Memory'][data['Expandable Memory'].isna() == True] = "No"

    data['Weight'] = data['Weight'].apply(for_integer)
    data['Weight'] = data['Weight'].fillna(data['Weight'].mean())
    data['Weight'] = data['Weight'].astype(int)

    data['SIM 2'] = data['SIM 2'].apply(for_string)
    data['SIM 2'][data['SIM 2'].isna() == True] = "Other"

    return data


# In[9]:


train = data_clean_2(train)
test = data_clean_2(test)

# check the number of features and data points in train
print("Number of data points in train: %d" % train.shape[0])
print("Number of features in train: %d" % train.shape[1])

# check the number of features and data points in test
print("Number of data points in test: %d" % test.shape[0])
print("Number of features in test: %d" % test.shape[1])


# Not very important feature
```

```
# In[10]:


def data_clean_3(x):

    data = x.copy()

    columns_to_remove = ['User Available Storage','SIM Size','Chipset','Processor','Autofocus','Aspect Ratio','Touch Screen',
                        'Bezel-less display','Operating System','SIM 1','USB Connectivity','Other Sensors','Graphics','FM Radio',
                        'NFC','Shooting Modes','Browser','Display Colour' ]

    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]


    columns_to_remove = [ 'Screen Resolution','User Replaceable','Camera Features',
                        'Thickness', 'Display Type']

    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]


    columns_to_remove = ['Fingerprint Sensor', 'Flash', 'Rating Count', 'Review Count','Image Resolution','Type','Expandable Memory',                        'Colours','Width','Model']
    columns_to_retain = list(set(data.columns)-set(columns_to_remove))
    data = data[columns_to_retain]

    return data


# In[11]:


train = data_clean_3(train)
test = data_clean_3(test)

# check the number of features and data points in train
print("Number of data points in train: %d" % train.shape[0])
print("Number of features in train: %d" % train.shape[1])

# check the number of features and data points in test
print("Number of data points in test: %d" % test.shape[0])
print("Number of features in test: %d" % test.shape[1])


# In[12]:


# one hot encoding

train_ids = train['PhoneId']
test_ids = test['PhoneId']

cols = list(test.columns)
cols.remove('PhoneId')
cols.insert(0, 'PhoneId')

combined = pd.concat([train.drop('Rating', axis=1)[cols], test[cols]])
```

```
print(combined.shape)
print(combined.columns)
#combined = combined.drop(['Brand'],axis=1)
combined = pd.get_dummies(combined)
print(combined.shape)
print(combined.columns)

train_new = combined[combined['PhoneId'].isin(train_ids)]
test_new = combined[combined['PhoneId'].isin(test_ids)]


# In[13]:


train_new = train_new.merge(train[['PhoneId', 'Rating']], on='PhoneId')


# In[14]:


# check the number of features and data points in train
print("Number of data points in train: %d" % train_new.shape[0])
print("Number of features in train: %d" % train_new.shape[1])

# check the number of features and data points in test
print("Number of data points in test: %d" % test_new.shape[0])
print("Number of features in test: %d" % test_new.shape[1])


# In[15]:


train_new.head()


# In[16]:


test_new.head()


# In[17]:


class Perceptron:
    def __init__ (self):
        self.w = None
        self.b = None
        self.device = None
        self.random_inits = {'normal':np.random.normal,'uniform':np.random.uniform}

    def model(self, x):
        dot_product = np.dot(self.w, x)
        return 1 if (dot_product >= self.b) else 0

    def predict(self, X):
        Y = []
        for x in X:
            result = self.model(x)
            Y.append(result)
```

```
        return np.array(Y)

    def fit(self, X, Y, epochs = 1, lr = 1,seeds=(1,1),init='normal',random_inits=True):
        if X.shape[0] != Y.shape[0]:
            print('X and Y have different shapes!',X.shape[0],'!=',Y.shape[0])
            return
        if random_inits:
            np.random.seed(seeds[0])
            self.w = self.random_inits[init](size=X.shape[1])
            np.random.seed(seeds[1])
            self.b = self.random_inits[init]()
        else:
            self.w = np.ones(X.shape[1])
            self.b = 0

        accuracy = {}
        max_accuracy = 0
        wt_matrix = []

        for i in range(epochs):
            for j in range(X.shape[0]):
                x,y = X[j],Y[j]
                y_pred = self.model(x)
                if y == 1 and y_pred == 0:
                    self.w = self.w + lr * x
                    self.b = self.b - lr * 1
                elif y == 0 and y_pred == 1:
                    self.w = self.w - lr * x
                    self.b = self.b + lr * 1

            accuracy[i] = accuracy_score(self.predict(X), Y)

            if (accuracy[i] > max_accuracy):
                max_accuracy = accuracy[i]
                chkptw = self.w
                chkptb = self.b

        self.w = chkptw
        self.b = chkptb

        print(max_accuracy)

        plt.plot(accuracy.values())
        plt.ylim([0, 1])
        plt.show()

        return max_accuracy


# In[18]:


class Perceptron_loss:
    def __init__ (self):
        self.w = None
        self.b = None
        self.device = None
        self.random_inits = {'normal':np.random.normal,'uniform':np.random.uniform}
        self.loss_function = [mean_squared_error,log_loss][0]
    def model(self, x):
```

```
        dot_product = np.dot(self.w, x)
        return 1 if (dot_product >= self.b) else 0

    def predict(self, X):
        Y = []
        for x in X:
            result = self.model(x)
            Y.append(result)
        return np.array(Y)

    def fit(self, X, Y, epochs = 1, lr = 1,seeds=(1,1),init='normal',random_inits=True):
        if X.shape[0] != Y.shape[0]:
            print('X and Y have different shapes!',X.shape[0],'!=',Y.shape[0])
            return
        if random_inits:
            np.random.seed(seeds[0])
            self.w = self.random_inits[init](size=X.shape[1])
            np.random.seed(seeds[1])
            self.b = self.random_inits[init]()
        else:
            self.w = np.ones(X.shape[1])
            self.b = 0

        accuracy = {}
        max_accuracy = 0
        wt_matrix = []
        loss = 1
        for i in range(epochs):
            for j in range(X.shape[0]):
                x,y = X[j],Y[j]
                y_pred = self.model(x)
                if y == 1 and y_pred == 0:
                    self.w = self.w + lr *loss* x
                    self.b = self.b - lr * 1
                elif y == 0 and y_pred == 1:
                    self.w = self.w - lr *loss* x
                    self.b = self.b + lr * 1
            pred = self.predict(X)
            accuracy[i] = accuracy_score(pred, Y)
            loss = self.loss_function(pred,Y)
            loss = -loss if  loss < 0 else loss

            if (accuracy[i] > max_accuracy):
                max_accuracy = accuracy[i]
                chkptw = self.w
                chkptb = self.b

        self.w = chkptw
        self.b = chkptb

        print(max_accuracy)

        plt.plot(accuracy.values())
        plt.ylim([0, 1])
        plt.show()

        return max_accuracy


# In[19]:
```

```
import torch
class Perceptron_cuda:
    def __init__ (self):
        self.w = None
        self.b = None
        self.device = None
        self.mt = 0
        self.random_inits = {'normal':np.random.normal,'uniform':np.random.uniform}

    def model(self, x):
        return torch.mm(x,self.w) >= self.b

    def predict(self, X):
        return self.model(X).cpu().numpy()

    def fit(self, X, Y, epochs = 1, lr = 1,seeds=(1,1),init='normal',random_inits=True):
        if X.shape[0] != Y.shape[0]:
            print('X and Y have different shapes!',X.shape[0],'!=',Y.shape[0])
            return
        if random_inits:
            np.random.seed(seeds[0])
            self.w = self.random_inits[init](size=X.shape[1])
            np.random.seed(seeds[1])
            self.b = self.random_inits[init]()
        else:
            self.w = np.ones(X.shape[1])
            self.b = 0

        accuracy = {}
        max_accuracy = 0
        wt_matrix = []
        lrc = lr
        zero,one = 0,1
        m1 = -1
        if torch.cuda.is_available():
            self.device = torch.device("cuda")
            self.w = torch.tensor(self.w,device=self.device).view(-1,1)
            self.b = torch.tensor(self.b,device=self.device,dtype=torch.double)
            X = torch.tensor(X,device=self.device)
            Y = torch.tensor(Y,device=self.device)
            lrc = torch.tensor(lrc,device=self.device,dtype=torch.double)
            zero = torch.tensor(0,device=self.device,dtype=torch.uint8)
            one = torch.tensor(1,device=self.device,dtype=torch.uint8)
            m1 = torch.tensor(-1,device=self.device,dtype=torch.uint8)
        for i in range(epochs):
            for j in range(X.shape[0]):
                x= X[j].view(-1,1)
                y = Y[j]
                y_pred = self.model(x.view(1,-1))
                if torch.sub(y,y_pred) == one:
                    self.w = torch.add(self.w, torch.mul(lrc, x))
                    self.b = torch.sub(self.b ,lrc)
                elif torch.sub(y,y_pred) == m1:
                    self.w = torch.sub(self.w, torch.mul(lrc, x))
                    self.b = torch.add(self.b ,lrc)

            accuracy[i] = accuracy_score(self.predict(X), Y.cpu().numpy())
            if (accuracy[i] > max_accuracy):
```

```
                max_accuracy = accuracy[i]
                chkptw = self.w
                chkptb = self.b

        self.w = chkptw
        self.b = chkptb

        print(max_accuracy)

        plt.plot(accuracy.values())
        plt.ylim([0, 1])
        plt.show()

        return max_accuracy


# In[20]:


perceptron = [Perceptron,Perceptron_cuda,Perceptron_loss][-1]()


# In[21]:


def
data_scale(data,scaler_class,cols_to_scale,drop_cols,scale_all=False,idcol='PhoneId',data_train_
scaler=None):
    scaler = scaler_class() if not data_train_scaler else data_train_scaler
    if scale_all:
        scaled = data.drop(drop_cols,axis=1)
        cols = scaled.columns
        if not data_train_scaler:
            scaler.fit(scaled)
        scaled[cols] = scaler.transform(scaled)
        data = scaled
    else:
        scaled,notscaled = data[cols_to_scale],data.drop(cols_to_scale,axis=1)
        if not data_train_scaler:
            scaler.fit(scaled)
        scaled[cols_to_scale] = scaler.transform(scaled)
        scaled[idcol] = data[idcol]
        data = scaled.merge(notscaled,on=idcol)
        data = data.drop(drop_cols,axis=1)
    return data,scaler


# In[22]:


scalers = [MinMaxScaler,StandardScaler]
test_ids = test_new[['PhoneId']]
scaler_class = scalers[1]
cols_to_scale = ['RAM','Internal Memory','Screen to Body Ratio (calculated)',
'Weight','Processor_frequency','Height', 'Capacity', 'Pixel Density', 'Screen Size', 'Resolution']
train_drop = ['PhoneId','Rating']
test_drop = ['PhoneId']
scale_all = False

x_train,train_scaler = data_scale(train_new,scaler_class,cols_to_scale,train_drop,scale_all)
```

```
x_test,test_scaler = data_scale(test_new,scaler_class,cols_to_scale,test_drop,scale_all,data_train_scaler=train_scaler)

y_train = np.array([ 1 if v>=THRESHOLD else 0 for v in train_new[['Rating']].values])
cols = x_train.columns
seeds = (1,1)


# In[23]:


def train(ep_lr):
    ep,lr = ep_lr
    print(ep,lr)
    return [ep,lr,pc.fit(x_train.values, y_train, ep,lr,seeds=seeds)]

def filtered(lst,func,index):
    maxval = func(lst[:,index])
    return np.array(list(filter(lambda v:v[index]==maxval,lst)))


# In[24]:


# parameters = list(ParameterGrid({'epochs':[1000,2000,5000,10000],'lr':[0.0001,0.001,0.01,0.1,1,0.2,0.02,0.3.0.03]}))
# parameters = list(ParameterGrid({'epochs':[100,1000,10000],'lr':[0.001,0.001,0.01,0.1,0.2,0.02,0.3,0.03]}))
# res = [train(tuple(k.values())) for k in parameters]


# In[25]:


# epi,lri,mai = 0,1,2 #indices
# best_params = np.array(res)
# best_params = filtered(best_params,max,mai)
# best_params = filtered(best_params,min,epi)
# #best_params = filtered(best_params,min,lri)
# best_params = best_params[0]
# print('Best hyperparameters found so far',best_params)


# In[26]:


#epochs,lr = int(best_params[epi]),best_params[lri]
epochs,lr,seeds = 1000,0.01,(0,0)
#epochs,lr,seeds = 100000,0.0001,(0,0)
print(epochs,lr)
max_accuracy = perceptron.fit(x_train.values, y_train, epochs, lr,seeds=seeds)


# In[27]:


# import time
# t1 = time.time()
# max_accuracy = pc.fit(x_train, y_train, 1000, 0.01,seeds=(2,4))
# print(time.time()-t1)
```

```
# In[28]:


plt.plot(perceptron.w)
plt.show()
weights = {}
for i in range(len(perceptron.w)):
    weights[cols[i]]=perceptron.w[i]
#sorted_by_value = {}
for c in list(sorted(weights, key=weights.get, reverse=True)):
    #sorted_by_value[c] = weights[c]
    print(c,weights[c])


# In[29]:


results = perceptron.predict(x_test.values)


# In[30]:


submission = pd.DataFrame({'PhoneId':test_ids['PhoneId'], 'Class':results})
submission = submission[['PhoneId', 'Class']]
submission.head()


# In[31]:


submission.to_csv("submission.csv", index=False)
```

Complete Code Github repo

https://github.com/AbhishekHupele/Infosys_Hackathon/upload

References:

https://www.kaggle.com/bittlingmayer/amazonreviews/kernels

https://padhai.onefourthlabs.in/courses/take/dl-feb-2019/lessons/6202710-excercises

https://www.researchgate.net/publication/325229033_Analyzing_Political_Sentiment_using_Twitter_Data

http://scikit-learn.org/stable/modules/model_evaluation.html

https://www.udemy.com/hands-on-natural-language-processing-using-python/