A DEEP DIVE INTO DECIPHERING AND DETECTING OFFENSIVE SPEECH OVER THE MICRO-BLOGGING WEBSITE - TWITTER

Submitted in partial fulfillment of the requirements for the degree

Bachelor of Technology

in

Computer Science and Engineering

by

LAKSHIT DUA- 18BCE0824

MOHAMMAD AMAN MIRZA- 18BCE0583

SHAIK HASEEB UR RAHMAN- 18BCE0646

Under the guidance of

Prof. Gopalakrishnan T

SCOPE

VIT, Vellore.

JUNE 2021

DECLARATION

We hereby declare that the thesis entitled "A deep dive into deciphering and detecting offensive speech over the micro-blogging website - Twitter" submitted by us, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by us under the supervision of Prof. Gopalakrishnan T.

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 2 June 2021

Signature of the Candidate

Lakshit Dua- 18BCE0824

Mohammad Aman Mirza- 18BCE0583

Shaik Haseeb Ur Rahman- 18BCE0646

CERTIFICATE

This is to certify that the thesis entitled "A deep dive into deciphering and detecting offensive speech over the micro-blogging website - Twitter" submitted by Lakshit Dua 18BCE0824, Mohammad Aman Mirza 18BCE0583 and Shaik Haseeb Ur Rahman 18BCE0646, SCOPE, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out under my supervision during the period 24.12.2020 to 08.06.2021, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 2 June 2021

Signature of the Guide

Internal Examiner **External Examiner**

Head of the Department

Computer Science and Engineering

ACKNOWLEDGEMENTS

I would like to extend my deepest thanks to Prof. Gopalakrishnan for providing me with this immensely generous opportunity as well as his guidance in the subject. I feel they were the factors that led me to successful completion of this project.

I would also like to extend my thanks to him for his time and effort in teaching us concepts with extreme rigor and passion. His efforts have resulted in the deep inculcation in of not only skills but a passion for the subject.

Lastly, I would like to thank my parents, friends and accomplices for their ever-lasting support in all my endeavours.

Lakshit Dua- 18BCE0824

Mohammad Aman Mirza- 18BCE0583

Shaik Haseeb Ur Rahman- 18BCE0646

Executive Summary

This project report can be summarised through a brief explanation of the data cleaning, pre-processing and training of classifier models utilized. The data, a set of 31,000 tweets sampled directly from twitter is first carried through the cleaning process. At this time, we try to eliminate roadblocks that may hamper proper training of classifier models at the last step, so we try to clean data using techniques like removing @user tags, stemming, lemmatisation, removal of stop words etc. This data is then sampled and pre-processing algorithms: standard TFIDF and our modification to TFIDF algorithm is applied and word weights are calculated. Five standard classification models are then trained on this data and the results i.e. accuracy of each model, are plotted on a graph and are compared.

TABLE OF CONTENTS

List of Figures

List of Abbreviations

TFIDF	Term Frequency-Inverse Document Frequency

1. INTRODUCTION

1.1. Objective

In our project, we aim to deliver a system using learning approach that automatically classifies tweets on Twitter into offensive speech and non-offensive. We will be using our own dataset extracted from twitter of around 31000+ tweets and will be perform data-cleaning procedure with the famous two algorithms namely, bag-of-words algorithm and the term frequency-inverse document frequency (TFIDF) values to machine learning model. Alongside, we will be improving on the TFIDF algorithm for its accuracy and efficacy pertaining to this very test case of hate and non-hate classification. Later, comparative analysis will be performed and will be studying the effectiveness of the models considering the accuracy of both of these approaches before and after improving the TFIDF into totally new algorithm. We will then train the models followed by making changes in them to provide best results. We will plot roc curve graphs and confusion matrices for each of the models for better understanding of the parameters and results of the graph will give us insight into the performance of the algorithms altogether.

1.2. Motivation

The 21st century has witnessed several developments in computing technology but none of them have come close enough to being as effective as machine learning and deep learning. With several advantages such as pattern detection, zero human intervention, continuous improvement, multi-dimensional, variety data support and flexible application, machine learning and deep learning has proved to be the most trendy and prominent technology there is today.

The internet has gone through a mass influx of end users that come from varied cultures and backgrounds. When cultures and backgrounds do collide at a common standing, the opinions turn out to be different leading to debates and verbal clashes when certain sensitive topics come up. There are around 500 million messages and posts created on twitter daily, therefore the job of filtering toxic content, offensive language and unsolicited opinions is a rather complex task.

This motivates us to take up this task and automate a classifier model of high accuracy that successfully predicts hate speech in a tweet before it is made official and raises alarm to block the offensive speech from ever making it to the internet. This will not only make twitter a more

appropriate and safer for work platform but also increase its credibility which on the current day is a place for the angry mob to belittle and judge every little opinion someone has.

1.3. Background

There have been two promising approaches for data pre-processing namely, Bag-of-Words algorithm and Term-Frequency Inverse-Document-Frequency (TFIDF). The bag-of-words approach is a simplified representation used in natural language processing in which a text like that of a sentence or a document is represented as the bag (multiset) of its words, discarding grammar and even word order while considering the multiplicity.

TFIDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection of words. It is used as a judging attribute in the aspects of information retrieval, text mining, and user modelling.

To reduce the option of error and to ensure maximum accuracy we will be running the cleaned data through five classification algorithms: Logistic Regression, Naive Bayes, Decision Tree, Random Forest.

The accuracy of these classifier models completely boils down to the training and testing data's domain and form. Considering the TFIDF approach, the score of a term among all tweets is given out based on their frequency but not their frequency in relevant places of relevant documents. This makes the system inefficient since the data stream is ultimately flawed with common stop words having greater weightage than other important ones. We have to just devise a system that accepts these shortcomings and makes up for them in a way that doesn't affect the ultimate goal of classification.

2. PROJECT DESCRIPTION AND GOALS

Our project proposes a system which classifies the tweets of twitter into hate and non-hate speech. Our system is basically a machine learning model which incorporates the two most famous algorithms Bag Of Words algorithm and the Term Frequency Inverse Document Frequency (TFIDF) which will be applied on the dataset which has been extracted earlier from twitter using a twitter extractor through twitter API. This dataset will constitute a total of 30,000 tweets. We also tried to improvise the Term Frequency Inverse Document Frequency (TFIDF)

by modifying the underlying formula of TDIDF which is basically product of number of occurrences of i and j with number of documents containing i and total number of documents. We amplified this product by multiplying the whole product with a factor which represents additional factor of hate level of each term or the non-hate level of terms to highlight more significant words that may occur. During this we will process, also work on other tasks of data cleaning such as removal of various Greek words, numeric values, hashtags, slangs and common words as this won't help on our analysis and unnecessary clutter our dataset. The pre-processing part will also incorporate basic tasks such as lowercasing the data to increase the consistency of the output, Stemming of the data is done in order to generate the stem of each word and replace it with the original and Noise removal. We will also analyze the profiling report of the dataset to check there is no missing elements and avoid duplicity of elements.

This will conclude our pre-processing part, now in the analysis part we will train this dataset using 5 most classification algorithm Naive Bayes, Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting and try to achieve a greater accuracy than the existing models. At last, we will conclude the project by achieving a greater accuracy and comparing all the classification algorithms through ROC curve and confusion matrix.

3. TECHNICAL SPECIFICATION

The entirety of the project code is written on Jupyter Notebook hosted on local server. The notebook runs on python 3.8 kernel that supports pip component library installer for additional processing components and classes available on the online pip library.

The following libraries were used in the code, including:

- nltk
- numpy
- pandas
- re
- matplotlib
- seaborn
- unidecode
- wordcloud

- operator
- pandas_profiling
- sklearn
- TFIDFvectorizer

The technical specifications of the project are very intricate and have to be extrapolated in a known vernacular to be understood with as much depth as they were first thought out. The significance of the term ‘modified TFIDF’ is immaculate. It signifies the modification of the existing TFIDF approach of text pre-processing.

Text preprocessing is the process of simplifying the task of analyses and predictability by converting the input data into a domain of better understanding for the machine. Some of the common tasks that are carried out in simple text preprocessing are:

- **Lowercasing** the data to increase the consistency of the output. Many times, in NLP applications such as smart lookups for word embedding, input variation occurs due to the presence of case disparity among similar words within the dataset. This is common for web mined data since it is a case sensitive operation and data tends to stick out of the general norm for case insensitiveness of input data to the smart lookup model for training and learning purposes. Mixed case data leads to inefficiency of the model and also obstructs the operation of effective learning of weights of each word. We carried out lowercasing for all the data that was input to the models during the initial data cleaning phase itself to save time in the future where other case sensitive operations were also to be performed.
- **Stemming** of the data is done in order to generate the stem of each word and replace it with the original. Any word in the English language excluding common nouns, pronouns, some adjectives etc. are made up of two parts; the stem and the ending. The stem is also referred to as the root of the word at times. This is most commonly seen in verbs. Let us consider the verb “to connect”. There are several conjugations of this verb: connected, connection, connects, connections etc. where all the forms, as you may have noticed have a common start to them i.e., connect. A common stemming algorithm is Porters Algorithm that uses heuristic processes to convert words into their root form. This results many times in words that do not make apparent sense but their presence is empirically important for the training and learning of the model. We carried out

stemming as a data cleaning operation. This was also done for the same reason as the task above, to avoid future of time wastage while carrying out case sensitive operations.

- **Lemmatization** is a similar operation to stemming but instead of heuristic approach, it takes a data driven approach towards the normalization and "stemming" of the words. It uses a premade database that acts as a dictionary to eliminate the ending part of words and yield proper mappings. Recent research has shown that it offers close to zero benefits over the most basic stemmer since the occurrence of words is important and not the form of word that occurs. Also, lemmatization is significantly slower that stemming for obvious reasons. Due to the downsides of lemmatization, we did not apply it to our data.
- **Noise Removal** is a very common data cleaning operation in web mining applications. Web mined data is generally in the form of raw html or some other markup that does not contain standard dictionary language or contains unprocessed syntax. To remove special characters while preserving the meaning behind the word in questions is referred to as noise removal. It is generally carried out right before the stemming operation since the result of stemming a noisy data may result in corrupt symbols and intangible other problems out of the scope of being removed with ease.

This was part of the operations that were carried out on the data to help boost accuracy in the final form of its processing.

After having successfully carried out these operations, the data was extracted and supplied to the next module that performed the learning and feature extraction part of the whole process.

Firstly, the data fed into this second-phase process was inspected and remaining potential roadblocks of inefficiency of the classifier model are eliminated. This includes the removal of extra columns in the data and the storing of individual words in separate lists. The TFIDF vectorized scores for each word are calculated. The 10,400 words that were used in this are features extracted from the tweet's dataset. Corpus, collection of tweets is generated which helps results in the vectorization of the scores of features, words, individually. Labelling of the dataset is used to calculate the occurrence of individual words in hate labeled and non-hate labeled tweets. Number of hate tweets a given term occurs in referred to as its hate level and similarly the number of non-hate tweets it occurs in is referred to as its non-hate level. The hate and non-hate level of each word is normalized and then multiplied to its TFIDF vectorized

value. This helps in enhancing the factor of occurrence of the word in a hate or non-hate word respectively.

4. DESIGN APPROACH AND DETAILS

As mentioned under the technical specifications section, the root of the project lies within the preprocessing of data to highlight the weights of words within the corpus or their TFIDF value, and hence enhancing the accuracy scores classifier models.

The overview of the design of the whole mechanism to classify the tweets is shown in the figure below.

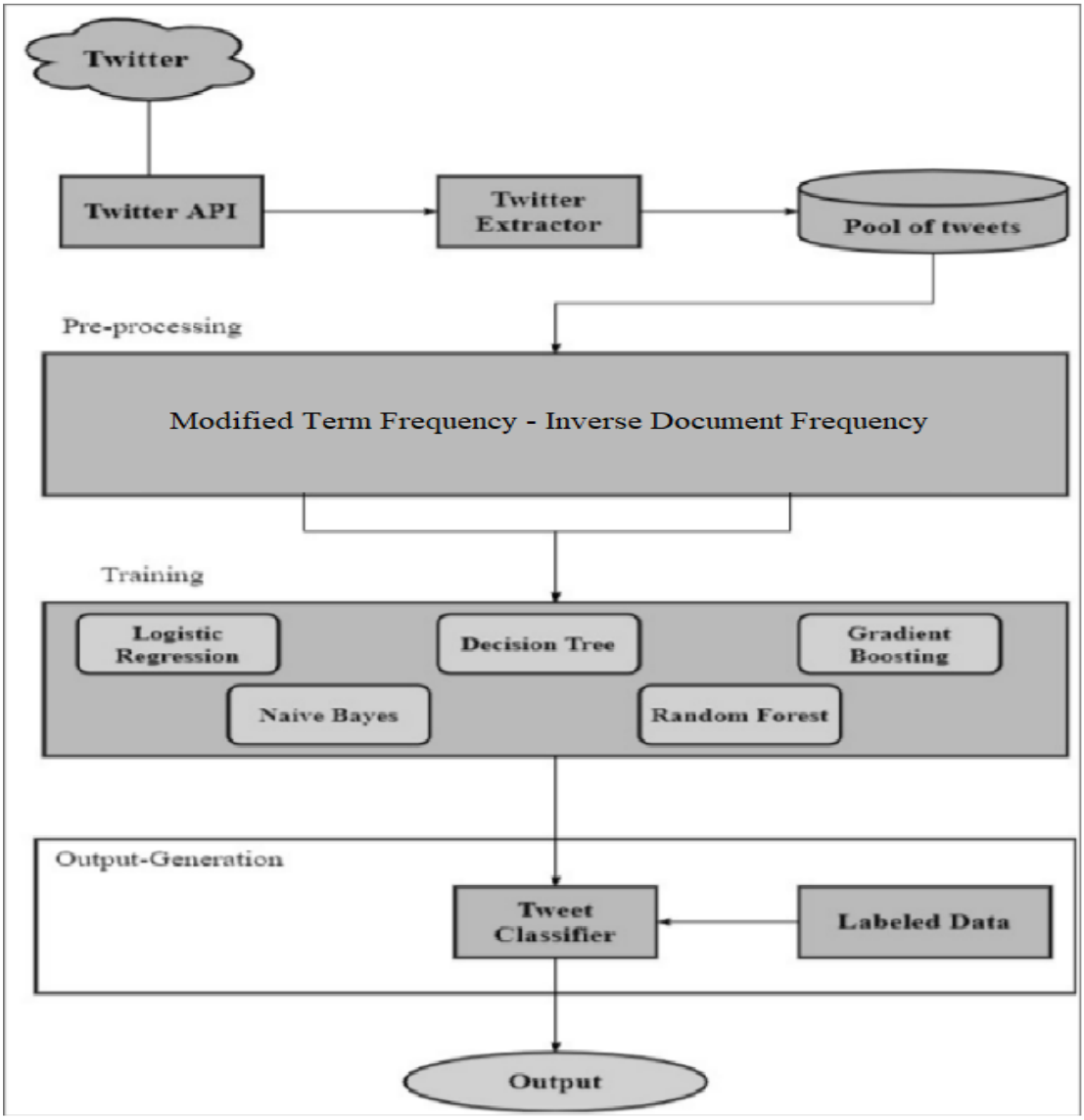


Figure 1: Design Diagram

4.1. Design Approach

The proposed preprocessing algorithm to replace vanilla TFIDF is designed with a result driven approach. TFIDF algorithm was designed to rank the occurrence/frequency of terms within a list of documents and it does so appropriately by factoring in two parameters for each term. These parameters, namely 'term frequency' and 'inverse document frequency' consider nothing but the weight of occurrence of the term individually and not in relation to any other factors that the term may potentially contribute to. This results in unexpected findings in many cases such as the one which we are discussing.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Figure 2: Original TFIDF formula to calculate weight of a term

The reason as to why original TFIDF fails to bring out concrete results in the present case can be identified by analyzing the method it follows. The term frequency of any term i in any document j is the number of times it occurs in that document and the inverse document frequency, the term that is utilized to normalize the weight, is the inverse of the number of documents that contain the term i (the inverse of the document frequency is multiplied by the total number of documents to normalize the value to ensure underfitting and log is taken to reduce the impact of term frequency on the whole equation).

This formula fails to highlight the weights of significant words that occur sparsely but carry high importance to the user's task at hand. As a direct consequence of the presence of the term frequency in the calculation, generally it is seen those words such as and, to, in, the etc. that fall under the stop word category attain the highest weights and uncommon ones the lowest.

Such consequences can be avoided by modifications to the algorithm as presented in this project. We can use the additional factor of hate level of each term or the non-hate level of terms to highlight more significant words that may occur sparsely. If a word such as 'kill' occurs 20 times total in the whole corpus and assuming it occurs in 5 documents out of 7000 total, the original TFIDF score of the term would come out to be something of the order of 63 (without normalization). Another word such as 'to' may occur in all 7000 documents and have a term frequency value in 5-digits will outweigh any other word in this calculation. This is where the original formula fails.

Let us take the modified TFIDF into account here in the same scenario. The word 'kill' may occur 20 times but out of those it will occur in hate labelled tweets and that contributes hate score of the word. Assuming it has a hate score of 20 and that is the max hate score of the corpus, it gets a 4-digit TFIDF value that is now much more comparable to the TFIDF score of a common word such as 'to', which leads to more accurate classification.

4.2. Codes and Standards

The process of classification of tweets is achieved by training 5 standard classifiers imported through the sklearn library. The standard models used are-

- **Logistic Regression –** Used for variables with binary dependencies to estimate logistic functions.
- **Decision Tree –** Uses a decision and consequence model and is generally applied to conditional control cases such as this.
- **Random Forest –** An exceptional case of decision trees for faster training times and better classification accuracy.
- **Gradient Boosting –** Collection of smaller decision tree like predictive units, used for classification and regression.
- **Naïve Bayes –** Probabilistic classifier model based on the Bayes' Theorem

4.3.Constraints, Alternatives and Tradeoffs

Modification to the TFIDF preprocessing mechanism makes it output better weights for each term based used for classification.

5. SCHEDULE, TASKS AND MILESTONES

Our project took 4 months to complete and was divided into 4 primary milestones which includes:

5.1. Extraction

In this part, we extracted tweets from twitter using a twitter extractor through twitter API. This dataset constituted a total of 30,000 tweets. This dataset was then passed for pre-processing.

5.2. Preprocessing

In this part, we first focused on cleaning the above extracted dataset through removal of various Greek words, numeric values, hashtags, slangs, and common words as this won't help on our analysis and unnecessary clutter our dataset. We also incorporated basic tasks such as lowercasing the data to increase the consistency of the output, Stemming of the data in order to generate the stem of each word and replace it with the original and Noise removal. We will also analyze the profiling report of the dataset to check there is no missing elements and avoid duplicity of elements. Then we applied the two algorithms Bag Of Words algorithm and the Term Frequency Inverse Document Frequency (TFIDF) and improvised the TFIDF on our clean dataset and this output was sent for further analysis. This part was done in a period two and half months.

5.3. Training, Analysis and Classification

In this part, we trained and tested our dataset using 5 most classification algorithm Naive Bayes, Logistic Regression, Decision Tree, Random Forest, and Gradient Boosting and tried to achieve a greater accuracy than the existing models. This was done in a matter of one month.

5.4. Output generation

Hence, in the last half month we concluded our project by achieving a greater accuracy and comparing all the classification algorithms through ROC curve and confusion matrix.

6. PROJECT DEMONSTRATION

Having acquired the unclean dataset, we need to import it into the code and apply code cleaning procedures to help with further process in the project. Data cleaning is done as follows –

In [1]:

```
import pandas_profiling
import nltk
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
import seaborn as sb
from nltk.corpus import stopwords
import warnings
warnings.filterwarnings("ignore")
import unidecode
from wordcloud import WordCloud
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
from nltk.stem import PorterStemmer
nltk.download('punkt')
from nltk.tokenize import word_tokenize
import matplotlib.animation as animation
import operator
import plotly.express as px
from collections import Counter
%matplotlib inline
```

```
[nltk_data] Downloading package wordnet to C:\Users\Techno
[nltk_data]     Luck\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\Techno
[nltk_data]     Luck\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [2]:

```
df = pd.read_csv('train_E6oV3lV.csv')
```

In [3]:

```
df.head()
```

In [6]:

```
df.shape
```

Out[6]:

```
(31962, 3)
```

In [7]:

```
df.drop_duplicates(inplace = True)
```

In [8]:

```
df.shape
```

Out[8]:

```
(31962, 3)
```

In [9]:

```
df['tweet'].isna().sum()
```

Out[9]:

```
0
```

In [10]:

```
df['label'].isna().sum()
```

Out[10]:

```
0
```

In [11]:

```
#Data doesn't contain duplicate values neither does it contain missing values
```

In [12]:

```
df.shape
```

Out[12]:

```
(31962, 3)
```

In [13]:

```
#Code to remove @
df['clean_tweet'] = df['tweet'].apply(lambda x : ' '.join([tweet for tweet in x.split()if not tweet.startswith("@")]))
```

In [14]:

```
df.head()
```

Out[14]:

	id	label	tweet	clean_tweet
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...

3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation

In [15]:

```
#Removing numbers
df['clean_tweet'] = df['clean_tweet'].apply(lambda x : ' '.join([tweet for tweet in x.sp
lit() if not tweet == '\d*']))
```

In [16]:

```
df.head()
```

Out[16]:

	id	label	tweet	clean_tweet
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...
2	3	0	bihday your majesty	bihday your majesty
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation

In [18]:

```
#Removing all the greek characters using unidecode library
df['clean_tweet'] = df['clean_tweet'].apply(lambda x : ' '.join([unidecode.unidecode(wor
d) for word in x.split()]))
```

In [19]:

```
df.head(10)
```

Out[19]:

	id	label	tweet	clean_tweet
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...
2	3	0	bihday your majesty	bihday your majesty
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before the...	[2/2] huge fan fare and big talking before the...
6	7	0	@user camping tomorrow @user @user @user @use...	camping tomorrow dannyal
7	8	0	the next school year is the year for exams.ð...	the next school year is the year for exams.d- ...
8	9	0	we won!!! love the land!!! #allin #cavs #champ...	we won!!! love the land!!! #allin #cavs #champ...
9	10	0	@user @user welcome here ! i'm it's so #gr...	welcome here ! i'm it's so #gr8 !

```
#To check the disappearance of greek symbols
df['clean_tweet'][7]
```

Out[20]:

```
"the next school year is the year for exams.d- can't think about that d #school #exams #h
ate #imagine #actorslife #revolutionschool #girl"
```

In [21]:

```
#Removing the word 'hmm' and it's variants
df['clean_tweet'] = df['clean_tweet'].apply(lambda x : ' '.join([word for word in x.spli
t() if not word == 'h(m)+' ]))
```

In [22]:

```
df.head()
```

Out[22]:

	id	label	tweet	clean_tweet
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...
2	3	0	bihday your majesty	bihday your majesty
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation

In [23]:

```
#Code for removing slang words
d = {'luv':'love','wud':'would','lyk':'like','wateva':'whatever','ttyl':'talk to you lat
er',
                'kul':'cool','fyn':'fine','omg':'oh my god!','fam':'family','bruh':'broth
er',
                'cud':'could','fud':'food'} ## Need a huge dictionary
words = "I luv myself"
words = words.split()
reformed = [d[word] if word in d else word for word in words]
reformed = " ".join(reformed)
```

In [24]:

```
reformed
```

Out[24]:

```
'I love myself'
```

In [25]:

```
df['clean_tweet'] = df['clean_tweet'].apply(lambda x : ' '.join(d[word] if word in d els
e word for word in x.split()))
```

In [26]:

```
df.head(30)
```

Out[26]:

	id	label	tweet	clean_tweet
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...
2	3	0	bihday your majesty	bihday your majesty
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...

In [27]:

```
#Finding words with # attached to it
df['#'] = df['clean_tweet'].apply(lambda x : ' '.join([word for word in x.split() if wor
d.startswith('#')]))
```

In [28]:

```
df.head()
```

Out[28]:

	id	label	tweet	clean_tweet	#
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...	#run
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...	#lyft #disapointed #getthanked
2	3	0	bihday your majesty	bihday your majesty	
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...	#model
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation	#motivation

In [29]:

```
frame = df['#']
```

In [38]:

```
data_frame = pd.concat([df,frame],axis = 1)
```

In [39]:

```
data_frame.head(10)
```

Out[39]:

	id	label	tweet	clean_tweet	#	Count(#)
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...	#run	#run
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...	#lyft #disapointed #getthanked	#lyft #disapointed #getthanked
2	3	0	bihday your majesty	bihday your majesty		No hashtags
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...	#model	#model
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation	#motivation	#motivation
5	6	0	[2/2] huge fan fare and big talking before the...	[2/2] huge fan fare and big talking before the...	#allshowandnogo	#allshowandnogo
6	7	0	@user camping tomorrow @user @user @user @use...	camping tomorrow dannyal		No hashtags
7	8	0	the next school year is the year for exams.ð...	the next school year is the year for exams.d- ...	#school #exams #hate #imagine #actorslife #rev...	#school #exams #hate #imagine #actorslife #rev...
8	9	0	we won!!! love the land!!! #allin #cavs #champ...	we won!!! love the land!!! #allin #cavs #champ...	#allin #cavs #champions #cleveland #clevelandc...	#allin #cavs #champions #cleveland #clevelandc...
9	10	0	@user @user welcome here ! i'm it's so #gr...	welcome here ! i'm it's so #gr8 !	#gr8	#gr8

In [40]:

```
data_frame.drop('#',axis = 1,inplace = True)
```

In [41]:

```
data_frame.head(10)
```

Out[41]:

In [45]:

```
#Removing stopwords
data_frame['clean_tweet'] = data_frame['clean_tweet'].apply(lambda x : ' '.join([word fo
r word in x.split() if not word in set(stopwords.words('english'))]))
```

In [46]:

```
data_frame.head()
```

Out[46]:

	id	label	tweet	clean_tweet	Hash words
0	1	0	@user when a father is dysfunctional and is s...	father dysfunctional selfish drags kids dysfun...	#run
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks #lyft credit can't use cause they whee...	#lyft #disapointed #getthanked
2	3	0	bihday your majesty	bihday majesty	No hashtags
3	4	0	#model i love u take with u all the time in ...	#model love u take u time urd+-!!! dddd dldldl	#model
4	5	0	factsguide: society now #motivation	factsguide: society #motivation	#motivation

In [47]:

```
df['clean_tweet_final'] = df['clean_tweet'].apply(lambda x : ' '.join([tweet for tweet i
n x.split()if not tweet.startswith("#")]))
```

In [48]:

```
df.head(30)
```

Out[48]:

	id	label	tweet	clean_tweet	#	clean_tweet_final
0	1	0	@user when a father is dysfunctional and is s...	when a father is dysfunctional and is so selfi...	#run	when a father is dysfunctional and is so selfi...
1	2	0	@user @user thanks for #lyft credit i can't us...	thanks for #lyft credit i can't use cause they...	#lyft #disapointed #getthanked	thanks for credit i can't use cause they don't...
2	3	0	bihday your majesty	bihday your majesty		bihday your majesty
3	4	0	#model i love u take with u all the time in ...	#model i love u take with u all the time in ur...	#model	i love u take with u all the time in urd+-!!! ...
4	5	0	factsguide: society now #motivation	factsguide: society now #motivation	#motivation	factsguide: society now
5	6	0	[2/2] huge fan fare and big talking before the...	[2/2] huge fan fare and big talking before the...	#allshowandnogo	[2/2] huge fan fare and big talking before the...
6	7	0	@user camping tomorrow @user @user @user @use...	camping tomorrow dannyal		camping tomorrow dannyal

21	22	0	sad little dude.. #badday #coneofshame #cats...	sad little dude.. #badday #coneofshame #cats #...	#badday #coneofshame #cats #pissed #funny #laughs	sad little dude..
22	23	0	product of the day: happy man #wine tool who'...	product of the day: happy man #wine tool who's...	#wine #weekend?	product of the day: happy man tool who's it's ...
23	24	1	@user @user lumpy says i am a . prove it lumpy.	lumpy says i am a . prove it lumpy.		lumpy says i am a . prove it lumpy.
24	25	0	@user #tgif #ff to my #gamedev #indiedev #i...	#tgif #ff to my #gamedev #indiedev #indiegamed...	#tgif #ff #gamedev #indiedev #indiegamedev #sq...	to my
25	26	0	beautiful sign by vendor 80 for $45.00!! #upsi...	beautiful sign by vendor 80 for $45.00!! #upsi...	#upsideofflorida #shopalyssas #love	beautiful sign by vendor 80 for $45.00!!
26	27	0	@user all #smiles when #media is !! ðð...	all #smiles when #media is !! dd #pressconfere...	#smiles #media #pressconference #antalya #turk...	all when is !! dd in ! sunday love! dda$?i,
27	28	0	we had a great panel on the mediatization of t...	we had a great panel on the mediatization of t...	#ica16	we had a great panel on the mediatization of t...
28	29	0	happy father's day @user ðððð	happy father's day dddd		happy father's day dddd
29	30	0	50 people went to nightclub to have a good nig...	50 people went to nightclub to have a good nig...	#rip#orlando	50 people went to nightclub to have a good nig...

In [ ]:

```
df.to_csv('clean_tweets_1.csv',index=False,columns=['label','clean_tweet_final'])
```

This completes the data cleaning process as we extract the clean_tweets column into new csv file and it is made ready to be used in the next module of preprocessing and model training.

In [2]:

```
import pandas as pd
```

In [3]:

```
df = pd.read_csv("clean_tweets_1.csv")
```

In [4]:

```
import numpy as np
```

In [6]:

```
df.head()
```

Out[6]:

	label	Clean_tweet
0	0	when a father is dysfunctional and is so selfi...
1	0	thanks for credit i can't use cause they don't...
2	0	bihday your majesty
3	0	i love u take with u all the time in urd+-!!! ...
4	0	factsguide: society now

In [7]:

```
df.shape
```

Out[7]:

```
(31962, 2)
```

In [8]:

```
dff = df.drop(['label'], axis = 1)
```

In [9]:

```
dff.head()
```

Out[9]:

	Clean_tweet
0	when a father is dysfunctional and is so selfi...
1	thanks for credit i can't use cause they don't...
2	bihday your majesty
3	i love u take with u all the time in urd+-!!! ...
4	factsguide: society now

```
from sklearn.model_selection import train_test_split
X_temp, X_test, y_temp, y_test = train_test_split(dff, list(df.label), test_size=0.1)
```

In [12]:

```
X_test.shape
```

Out[12]:

```
(3197, 1)
```

In [13]:

```
X_temp.shape
```

Out[13]:

```
(28765, 1)
```

In [14]:

```
len(y_test)
```

Out[14]:

```
3197
```

In [ ]:

```

```

In [15]:

```
type(y_temp)
```

Out[15]:

```
list
```

In [16]:

```
X_temp['label'] = y_temp
```

```
<ipython-input-16-696ff4fc08a7>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  X_temp['label'] = y_temp
```

In [17]:

```
X_temp.head()
```

Out[17]:

In [20]:

```
nonhate = X_temp[X_temp['label'] == 0]
```

In [21]:

```
nonhate.head()
```

Out[21]:

	Clean_tweet	label
23685	nice inseion	0
4216	bihday mom...#bihday	0
20621	cant wait for nxt thurs. pay .. coppers.. d1d,...	0
10670	the first mammal known to be wiped out by the ...	0
9204	if canA't u happy !!! uA'll b	0

In [22]:

```
hate = X_temp[X_temp.label == 1]
```

In [23]:

```
hate.shape[0]
```

Out[23]:

```
1994
```

In [24]:

```
nonhatesample = nonhate.sample(n = hate.shape[0])
```

In [25]:

```
nonhatesample.head()
```

Out[25]:

	Clean_tweet	label
20348	dad's happy cam... gbp 16.00 get here:	0
8888	a economy to growth just 0.3%; boj likely to e...	0
17373	i am funny.	0
3253	me haces feliz. d davidcusguen al	0
16816	dining out d' moonlight beach	0

In [44]:

```
corpus = []
for i in range(ds.shape[0]):
    corpus.append(ds.iloc[i][0])
```

In [45]:

```
corpus
```

Out[45]:

```
["why do refer to everyone else as racist. simple, they're projecting their intolerance."
,
 "like to know why you can't buy pot at the local drug store? marijuana unleashed at iboo
ks. a|",
 'you might be a libtard if...',
 '.@user my gf used uber without forcing language preferences so you are reading my data
and making choices',
 'not all muslims are islamic extremists (very, very few are).',
 "is still rooted in our society's attitude towards black women's hair. read more from ca
ndide uyanze:a|",
 'is not a problem in america today. is. dumb is an equal oppounity killer of dreams.',
 "i'm guessing a shiless man doesn't objectify himself by being shiless, but a shiless wo
man is?",
 'more evil jew why is it jew controlled media has never and never will cite any non whit
e groua|',
 'jokes about women penis humping girl',
 '"kevin didnat understand what head done, but he knew head fucked up." by',
 'this obviously gets on &amp;amp; maybe two seconds after boarding - thinks he "knows" whata
|',
 "can't effo a logic point w/out or especially if that's wha|",
 'american taxpayers have also lost their homes by the millions a| a|',
 'no comment! in',
```

Using original TFIDF vectorizer to produce weights for terms. We use this TFIDF weight to compare the accuracy of model trained using TFIDF and modified TFIDF and compare the accuracy scores.

In [48]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [49]:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_corpus)
feature_names = vectorizer.get_feature_names()
dense = X.todense()
denselist = dense.tolist()
df2 = pd.DataFrame(denselist, columns=feature_names)
df2
```

Out[49]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zine	zionism	zit	zomg	zone	zoro	zum
0	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7162	0.262595	0.0	0.0	0.30162	0.0	0.0	0.0	0.0	0.538143	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7163	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7164	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [51]:

```
tdf = df2
tdf['labelxyz'] = list(ds[0:len(tdf)].label)
```

In [52]:

```
tdf.tail()
```

Out[52]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zionism	zit	zomg	zone	zoro	zuma	zur
7162	0.262595	0.0	0.0	0.30162	0.0	0.0	0.0	0.0	0.538143	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
7163	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
7164	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
7165	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
7166	0.000000	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 10243 columns

In [53]:

```
print(tdf[tdf.labelxyz == 1])
```

```
     00  000  000001   01  039  04pm   05  05pm   06  0618a  ...  zionism  \
0   0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...      0.0
1   0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...      0.0
2   0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...      0.0
3   0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...      0.0
4   0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...      0.0
```

In [54]:

```
tdf_hate = tdf[tdf.labelxyz == 1]
tdf_hate_new = tdf_hate
tdf_hate_new=tdf_hate_new.drop("labelxyz",axis=1)
print(tdf_hate_new)
```

```
       00  000  000001   01  039  04pm   05  05pm   06  0618a  ...  zine  \
0     0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
1     0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
2     0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
3     0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
4     0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
...   ...  ...     ...  ...  ...   ...  ...   ...  ...    ...  ...   ...
7124  0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
7134  0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
7145  0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
7151  0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0
7166  0.0  0.0     0.0  0.0  0.0   0.0  0.0   0.0  0.0    0.0  ...   0.0

      zionism  zit  zomg  zone  zoro  zuma  zurich  zydeco   zz
0         0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
1         0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
2         0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
3         0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
4         0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
...       ...  ...   ...   ...   ...   ...     ...     ...  ...
7124      0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
7134      0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
7145      0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
7151      0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0
7166      0.0  0.0   0.0   0.0   0.0   0.0     0.0     0.0  0.0

[2240 rows x 10242 columns]
```

In [55]:

```
tdf_hate.shape
```

Out[55]:

```
(2240, 10243)
```

In [56]:

```
tdf_nonhate = tdf[tdf.labelxyz == 0]
```

In [57]:

```
tdf_nonhate_new = tdf_nonhate
```

In [58]:

```
tdf_nonhate.shape
```

Out[58]:

```
(4927, 10243)
```

In [59]:

```
X_train_hate = tdf_hate.sample(frac=0.9, random_state=0)

X_test_hate = tdf_hate.drop(X_train_hate.index)

X_train_nonhate =  tdf_nonhate.sample(frac=0.406, random_state=0)

X_test_nonhate = tdf_nonhate.drop(X_train_nonhate.index)
```

In [60]:

```
X_train_df = pd.concat([X_train_hate, X_train_nonhate], axis = 0)
X_train_df
```

Out[60]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zionism	zit	zomg	zone	zoro	zuma	zurich	zydeco	zz
670	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
570	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1909	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7099	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5392	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3949	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4542	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [61]:

```
X_train = X_train_df.drop(['labelxyz'], axis = 1)

y_train = list(X_train_df.labelxyz)

X_test_df = pd.concat([X_test_hate, X_test_nonhate], axis = 0)

X_test = X_test_df.drop(['labelxyz'], axis = 1)

y_test = list(X_test_df.labelxyz)
```

In [62]:

```
#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(df2, list(ds.label), test_size=0.2)
```

In [63]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

In [64]:

```
# Use Cross-validation.
from sklearn.model_selection import cross_val_score

# Logistic Regression
log_reg = LogisticRegression()
log_scores = cross_val_score(log_reg, X_train, y_train, cv=3)
log_reg_mean = log_scores.mean()


# Naives Bayes
nav_clf = GaussianNB()
nav_scores = cross_val_score(nav_clf, X_train, y_train, cv=3)
nav_mean = nav_scores.mean()

# Create a Dataframe with the results.
d = {'Classifiers': ['Logistic Reg.', 'Naives Bayes'],
     'Crossval Mean Scores': [log_reg_mean, nav_mean]}

result_df = pd.DataFrame(data=d)

result_df
```

Out[64]:

	Classifiers	Crossval Mean Scores
0	Logistic Reg.	0.736058
1	Naives Bayes	0.654885

In [65]:

```
from sklearn.metrics import accuracy_score
nav_clf = GaussianNB()
nav_clf.fit(X_train, y_train)
predict_nav = nav_clf.predict(X_test)
accuracy_score(y_test, predict_nav)
```

Out[65]:

0.7140590288797207

In [66]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict_nav)
```

Out[66]:

```
array([[2095,  832],
       [  69,  155]], dtype=int64)
```

In [67]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
predict_log = log_reg.predict(X_test)
accuracy_score(y_test, predict_log)
```

Out[67]:

0.7676927959377975

In [68]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict_log)
```

Out[68]:

```
array([[2250,  677],
       [  55,  169]], dtype=int64)
```

In [73]:

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier()
rand_clf.fit(X_train, y_train)
predict = rand_clf.predict(X_test)
accuracy_score(y_test, predict)
```

Out[73]:

0.7829260552205649

In [74]:

```
confusion_matrix(y_test, predict)
```

Out[74]:

```
array([[2313,  614],
       [  70,  154]], dtype=int64)
```

In [75]:

```
189/(189+35)
```

Out[75]:

0.84375

```
from sklearn import tree
dt_clf = tree.DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
predict_dt = dt_clf.predict(X_test)
accuracy_score(y_test, predict_dt)
```

Out[76]:

```
0.6883529038400508
```

In [77]:

```
confusion_matrix(y_test, predict_dt)
```

Out[77]:

```
array([[2018,  909],
       [  73,  151]], dtype=int64)
```

In [78]:

```
195/224
```

Out[78]:

```
0.8705357142857143
```

In [79]:

```
from sklearn.ensemble import GradientBoostingClassifier
grad_clf = GradientBoostingClassifier()
grad_clf.fit(X_train, y_train)
predict_grad = grad_clf.predict(X_test)
accuracy_score(y_test, predict_grad)
```

Out[79]:

```
0.7619803237067597
```

In [80]:

```
confusion_matrix(y_test, predict_grad)
```

Out[80]:

```
array([[2267,  660],
       [  90,  134]], dtype=int64)
```

In [81]:

```
144/224
```

Out[81]:

```
0.6428571428571429
```

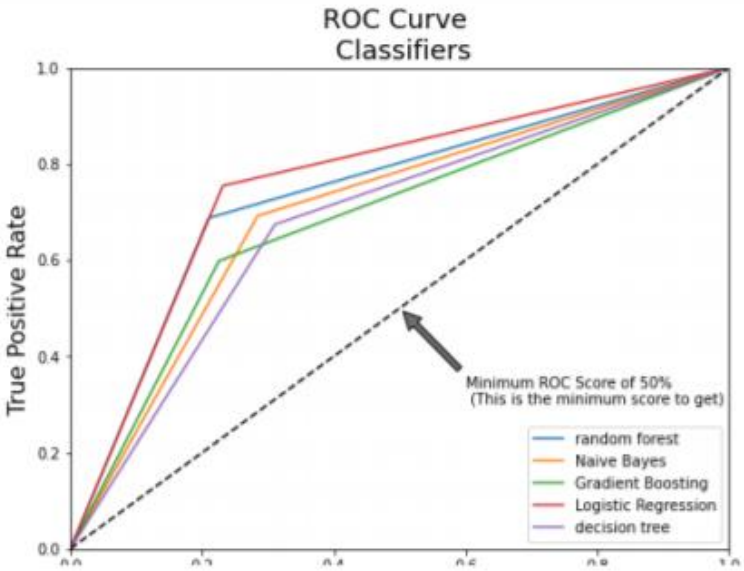
****

```
Logistic Regression Score:  0.7615847222900093
Naive Bayes Score:  0.7038571001513008
Random Forest Score:  0.7388644516569867
Decision Tree Score:  0.6817751293376934
Grad Boosting Score:  0.6863637195568353
```

In [85]:

```
def graph_roc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log
_fpr, log_tpr, dt_fpr, dt_tpr):
    plt.figure(figsize=(8,6))
    plt.title('ROC Curve \n Classifiers', fontsize=18)
    plt.plot(rand_fpr, rand_tpr, label='random forest')
    plt.plot(nav_fpr, nav_tpr, label='Naive Bayes')
    plt.plot(grad_fpr, grad_tpr, label='Gradient Boosting')
    plt.plot(log_fpr, log_tpr, label='Logistic Regression')
    plt.plot(dt_fpr, dt_tpr, label='decision tree')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.
5, 0.5), xytext=(0.6, 0.3),
                 arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                 )
    plt.legend()

graph_roc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log_fpr
, log_tpr, dt_fpr, dt_tpr)
plt.show()
```



These are the accuracy scores and the roc curve for the original TFIDF preprocessed models.

Now starting with the modified TFIDF algorithm, we first extract the words –

In [86]:

```
words = []
for k in range(len(cleaned_corpus)):
    tweet_k = cleaned_corpus[k].split(" ")
    for m in range(len(tweet_k)):
        words.append(tweet_k[m])
print(words)
from collections import Counter
counter_obj = Counter(words)
top100 = counter_obj.most_common(1000)
top100words = []
for i in range(len(top100)):
    top100words.append(top100[i][0])
```

```
['why', 'do', 'refer', 'to', 'everyone', 'else', 'as', 'racist.', 'simple,', "they're", '
projecting', 'their', 'intolerance.', 'like', 'to', 'know', 'why', 'you', "can't", 'buy',
'pot', 'at', 'the', 'local', 'drug', 'store?', 'marijuana', 'unleashed', 'at', 'ibooks.',
'a|', 'you', 'might', 'be', 'a', 'libtard', 'if...', '.@user', 'my', 'gf', 'used', 'uber'
, 'without', 'forcing', 'language', 'preferences', 'so', 'you', 'are', 'reading', 'my', '
data', 'and', 'making', 'choices', 'not', 'all', 'muslims', 'are', 'islamic', 'extremists
', '(very,', 'very', 'few', 'are).', 'is', 'still', 'rooted', 'in', 'our', "society's", '
attitude', 'towards', 'black', "women's", 'hair.', 'read', 'more', 'from', 'candide', 'uy
anze:a|', 'is', 'not', 'a', 'problem', 'in', 'america', 'today.', 'is.', 'dumb', 'is', 'a
n', 'equal', 'oppounity', 'killer', 'of', 'dreams.', "i'm", 'guessing', 'a', 'shiless', '
man', "doesn't", 'objectify', 'himself', 'by', 'being', 'shiless,', 'but', 'a', 'shiless'
, 'woman', 'is?', 'more', 'evil', 'jew', 'why', 'is', 'it', 'jew', 'controlled', 'media',
'has', 'never', 'and', 'never', 'will', 'cite', 'any', 'non', 'white', 'groua|', 'jokes',
'about', 'women', 'penis', 'humping', 'girl', '"kevin', 'didnat', 'understand', 'what', '
head', 'done,', 'but', 'he', 'knew', 'head', 'fucked', 'up."', 'by', 'this', 'obviously',
'gets', 'on', '&amp;', 'maybe', 'two', 'seconds', 'after', 'boarding', '-', 'thinks', 'he
', '"knows"', 'whata|', "can't", 'effo', 'a', 'logic', 'point', 'w/out', 'or', 'especiall
y', 'if', "that's", 'wha|', 'american', 'taxpayers', 'have', 'also', 'lost', 'their', 'ho
mes', 'by', 'the', 'millions', 'a|', 'a|', 'no', 'comment!', 'in', 'spos', 'illustrated',
'girls', 'naked', 'sex', 'videos', 'great', 'aicle', 'of', 'truth', 'about', 'needs.', 'c
omments', 'back', 'up', 'why', "it's", 'needed.', '-', 'via', 'so,', 'what', 'are', 'you'
, 'saying,', 'blacks', '=', 'criminals?', 'presenter', 'to', 'asian', 'woman', 'in', 'edi
nburgh:', '"what', 'brings', 'you', 'here?"', 'asian', 'woman:', '"i', 'live', 'here."',
'...', 'how', 'would', 'you', 'promote', '...', 'hitting', 'on', 'women', '...', 'condoni
ng', 'assault', '...', '&amp;', '...', 'anda|', "i'm", 'always', 'amazed', 'at', 'the', '
```

Calculating the word_hate_level for each of the words -

In [87]:

```
word_hate_level = {}
x=0
for all_words in feature_names:
    word_hate_level[all_words] = 0
hate_freqs = tdf_hate_new.values
print(len(hate_freqs)," ",hate_freqs[0:5]," ")
for hate_freq in hate_freqs:
    float_freq = hate_freq.tolist()
    x=0
    for hate_word_freq in float_freq:
        if hate_word_freq != 0:
            #print(hate_word_freq," ",x)
            try:
                word_hate_level[feature_names[x]]+=1
            except IndexError:
                print(feature_names[x]," ",hate_word_freq," ",x)
        x=x+1
#print(word_hate_level)
max_hate_key = max(word_hate_level,key=word_hate_level.get)
#word_hate_level[max_hate_key]=0
#max_hate_key = max(word_hate_level,key=word_hate_level.get)
print(word_hate_level[max_hate_key])
```

```
2240   [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
677
```

Similarly calculating non-hate-level for each word -

In [88]:

```
word_nonhate_level = {}
y=0
for all_words in feature_names:
    word_nonhate_level[all_words] = 0
nonhate_freqs = tdf_nonhate_new.values
for nonhate_freq in nonhate_freqs:
    nfloat_freq = nonhate_freq.tolist()
    y=0
    for nonhate_word_freq in nfloat_freq:
        if nonhate_word_freq != 0:
            #print(nonhate_word_freq," ",y)
            try:
                word_nonhate_level[feature_names[y]]+=1
            except IndexError:
                print(feature_names[y]," ",nonhate_word_freq," ",y)
        y=y+1
```

Appending the normalized hate and non-hate values to the original data frames by multiplying into non-zero values –

In [89]:

```
hate_keys, hate_values = zip(*word_hate_level.items())
nonhate_keys, nonhate_values = zip(*word_nonhate_level.items())
hate_values = list(hate_values)
nonhate_values = list(nonhate_values)
print(len(hate_keys)," ",len(hate_freqs))
temp_vals_full = []
for hate_freq in hate_freqs:
    float_freq_new = hate_freq.tolist()
    x=0
    temp_vals = []
    for hate_word_freq_new in float_freq_new:
        temp_vals.append(hate_word_freq_new*(float(hate_values[x])/float(word_hate_level
[max_hate_key])))
        x+=1
    temp_vals_full.append(temp_vals)
print(len(temp_vals_full))
```

```
10242   2240
2240
```

In [90]:

```
hate_df = pd.DataFrame(temp_vals_full, columns=feature_names)
hate_df
```

Out[90]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zine	zionism	zit	zomg	zone	zoro	zuma	zurich	zydec
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2235	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2236	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2237	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

In [91]:

```
print(len(nonhate_keys)," ",len(nonhate_freqs))
temp_vals_full = []
for nonhate_freq in nonhate_freqs:
    nfloat_freq_new = nonhate_freq.tolist()
    y=0
    temp_vals = []
    for nonhate_word_freq_new in nfloat_freq_new:
        temp_vals.append(nonhate_word_freq_new*(float(nonhate_values[y])/float(word_nonh
ate_level[max_nonhate_key])))
        y+=1
    temp_vals_full.append(temp_vals)
print(len(temp_vals_full))
```

```
10242   4927
4927
```

In [92]:

```
nonhate_df = pd.DataFrame(temp_vals_full, columns=feature_names)
nonhate_df
```

Out[92]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zine	zionism	zit	zomg	zone	zoro	zun
0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
1	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Training the models with modified TFIDF values -

In [93]:

```
X_train_hate_new = hate_df.sample(frac=0.9, random_state=0)

X_test_hate_new = hate_df.drop(X_train_hate_new.index)

X_train_nonhate_new =  nonhate_df.sample(frac=0.406, random_state=0)

X_test_nonhate_new = nonhate_df.drop(X_train_nonhate_new.index)

X_train_df_new = pd.concat([X_train_hate_new, X_train_nonhate_new], axis = 0)
X_train_df_new
```

Out[93]:

	00	000	000001	01	039	04pm	05	05pm	06	0618a	...	zine	zionism	zit	zomg	zone	zoro	zuma	zurich	zydec
670	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
570	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1909	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2231	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

```
X_train_new = X_train_df_new

y_train = list(X_train_df.labelxyz)

X_test_df_new = pd.concat([X_test_hate_new, X_test_nonhate_new], axis = 0)

X_test_new = X_test_df_new

y_test = list(X_test_df.labelxyz)
```

In [98]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

In [99]:

```
# Use Cross-validation.
from sklearn.model_selection import cross_val_score

# Logistic Regression
log_reg = LogisticRegression()
log_scores = cross_val_score(log_reg, X_train_new, y_train, cv=3)
log_reg_mean = log_scores.mean()


# Naives Bayes
nav_clf = GaussianNB()
nav_scores = cross_val_score(nav_clf, X_train_new, y_train, cv=3)
nav_mean = nav_scores.mean()

# Create a Dataframe with the results.
d = {'Classifiers': ['Logistic Reg.', 'Naives Bayes'],
    'Crossval Mean Scores': [log_reg_mean, nav_mean]}

result_df = pd.DataFrame(data=d)

result_df
```

Out[99]:

	Classifiers	Crossval Mean Scores
0	Logistic Reg.	0.648910
1	Naives Bayes	0.783862

Accuracy scores of the models -

In [100]:

```
from sklearn.metrics import accuracy_score
nav_clf = GaussianNB()
nav_clf.fit(X_train_new, y_train)
predict_nav = nav_clf.predict(X_test_new)
accuracy_score(y_test, predict_nav)
```

Out[100]:

```
0.8460806093303713
```

In [101]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict_nav)
```

Out[101]:

```
array([[2486,  441],
       [  44,  180]], dtype=int64)
```

In [102]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train_new, y_train)
predict_log = log_reg.predict(X_test_new)
accuracy_score(y_test, predict_log)
```

Out[102]:

```
0.7238971754998413
```

In [103]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict_log)
```

Out[103]:

```
array([[2144,  783],
       [  87,  137]], dtype=int64)
```

In [104]:

```
from sklearn.ensemble import RandomForestClassifier
rand_clf = RandomForestClassifier()
rand_clf.fit(X_train_new, y_train)
predict = rand_clf.predict(X_test)
accuracy_score(y_test, predict)
```

Out[104]:

```
0.8060933037131O7
```

In [105]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict)
```

Out[105]:

```
array([[2402,  525],
       [  86,  138]], dtype=int64)
```

In [106]:

```
from sklearn import tree
dt_clf = tree.DecisionTreeClassifier()
dt_clf.fit(X_train_new, y_train)
predict_dt = dt_clf.predict(X_test_new)
accuracy_score(y_test, predict_dt)
```

Out[106]:

```
0.8181529673119644
```

In [107]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predict_dt)
```

Out[107]:

```
array([[2394,  533],
       [  40,  184]], dtype=int64)
```

ROC accuracy scores –

ROC CURVE FOR MODIFIED TFIDF

In [109]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

print('Logistic Regression Score: ', roc_auc_score(y_test, p))
print('Naive Bayes Score: ', roc_auc_score(y_test, predict_nav))
print('Random Forest Score: ', roc_auc_score(y_test, predict))
print('Decision Tree Score: ', roc_auc_score(y_test, predict_dt))
print('Grad Boosting Score: ', roc_auc_score(y_test, predict_grad))
```

```
Logistic Regression Score:  0.7615847222900093
Naive Bayes Score:  0.8264526087168724
Random Forest Score:  0.7183534457513789
Decision Tree Score:  0.8196654302308556
Grad Boosting Score:  0.8196646676265312
```

In [110]:

```
from sklearn.metrics import roc_curve

log_fpr, log_tpr, threshold = roc_curve(y_test,p)
nav_fpr, nav_tpr, threshold = roc_curve(y_test, predict_nav)
rand_fpr, rand_tpr, thresold = roc_curve(y_test, predict)
dt_fpr, dt_tpr, threshold = roc_curve(y_test, predict_dt)
grad_fpr, grad_tpr, threshold = roc_curve(y_test, predict_grad)

#nav_fpr, nav_tpr, nav_threshold = roc_curve(y_train, naives_y_scores)
```

In [111]:

```
def graph_roc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log
_fpr, log_tpr, dt_fpr, dt_tpr):
    plt.figure(figsize=(8,6))
    plt.title('ROC Curve \n Classifiers', fontsize=18)
    plt.plot(rand_fpr, rand_tpr, label='random forest')
    plt.plot(nav_fpr, nav_tpr, label='Naive Bayes')
    plt.plot(grad_fpr, grad_tpr, label='Gradient Boosting')
    plt.plot(log_fpr, log_tpr, label='Logistic Regression')
    plt.plot(dt_fpr, dt_tpr, label='decision tree')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
    plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score to get)', xy=(0.
5, 0.5), xytext=(0.6, 0.3),
                 arrowprops=dict(facecolor='#6E726D', shrink=0.05),
                 )
    plt.legend()

graph_roc_curve_multiple(rand_fpr, rand_tpr, nav_fpr, nav_tpr,grad_fpr, grad_tpr, log_fpr
, log_tpr, dt_fpr, dt_tpr)
plt.show()
```

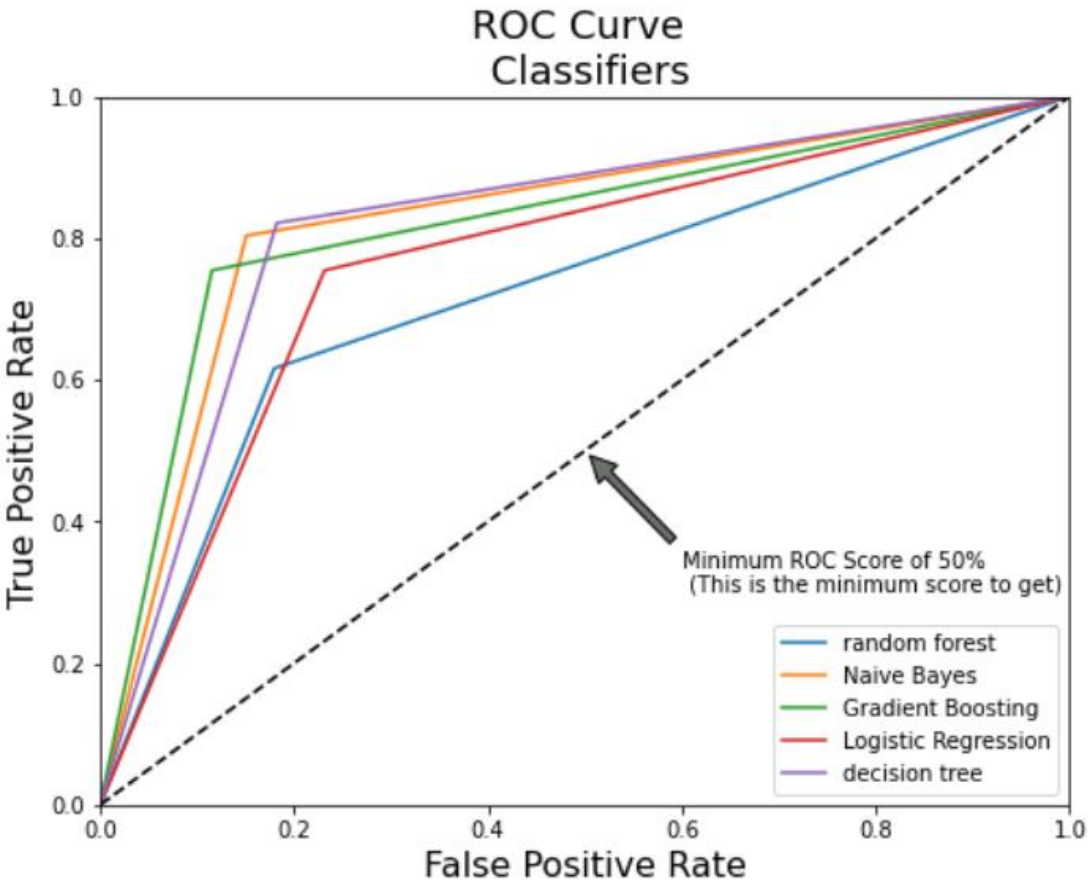


Figure 3: ROC Curve Classifiers

7. RESULTS & DISCUSSIONS

Original TFIDF results -

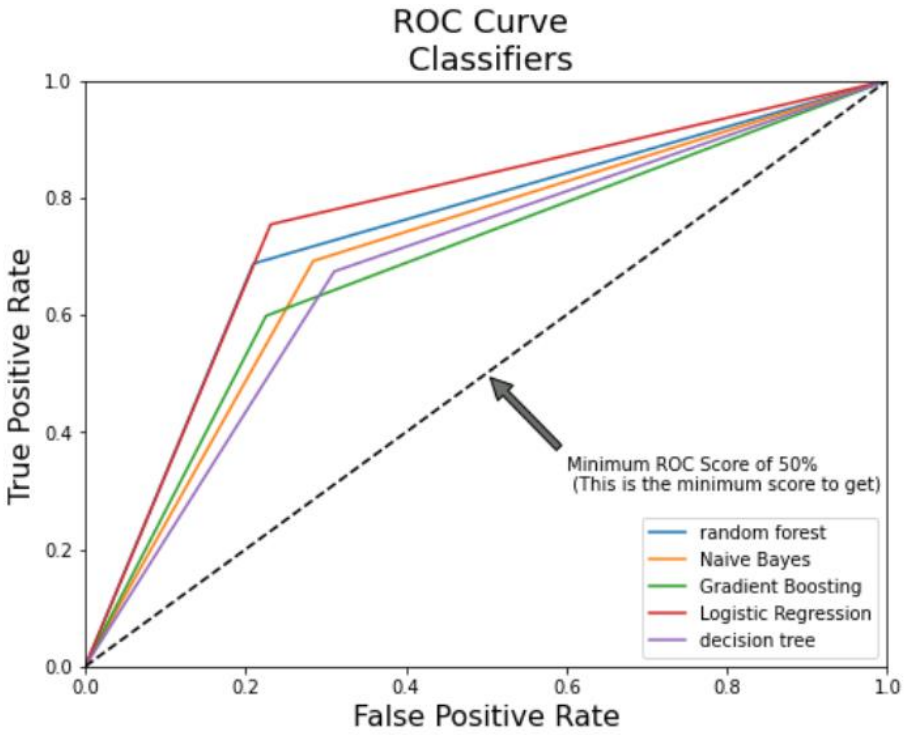


Figure 4: ROC Curve Classifiers of Original TFIDF

Best accuracy achieved through random forest at 78% -

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier()
rand_clf.fit(X_train, y_train)
predict = rand_clf.predict(X_test)
accuracy_score(y_test, predict)
```

```
0.7829260552205649
```

Figure 5: Accuracy of Random Forest

ROC Scores of all the models -

```
Logistic Regression Score:  0.7615847222900093
Naive Bayes Score:  0.7038571001513008
Random Forest Score:  0.7388644516569867
Decision Tree Score:  0.6817751293376934
Grad Boosting Score:  0.6863637195568353
```

Figure 6: ROC Scores of all models

Highest accuracy score with modified TFIDF at 87% for gradient boosting + the confusion matrix -

```
gradient boosting
0.875595049190733
[[2590  337]
 [  55  169]]
```

Figure 7: Gradient Boosting Accuracy Score of Modified TFIDF

ROC scores of all models -

```
Logistic Regression Score:  0.7615847222900093
Naive Bayes Score:  0.8264526087168724
Random Forest Score:  0.7183534457513789
Decision Tree Score:  0.8196654302308556
Grad Boosting Score:  0.8196646676265312
```

Figure 8: ROC Scores of all the Models

ROC curve for modified TFIDF -

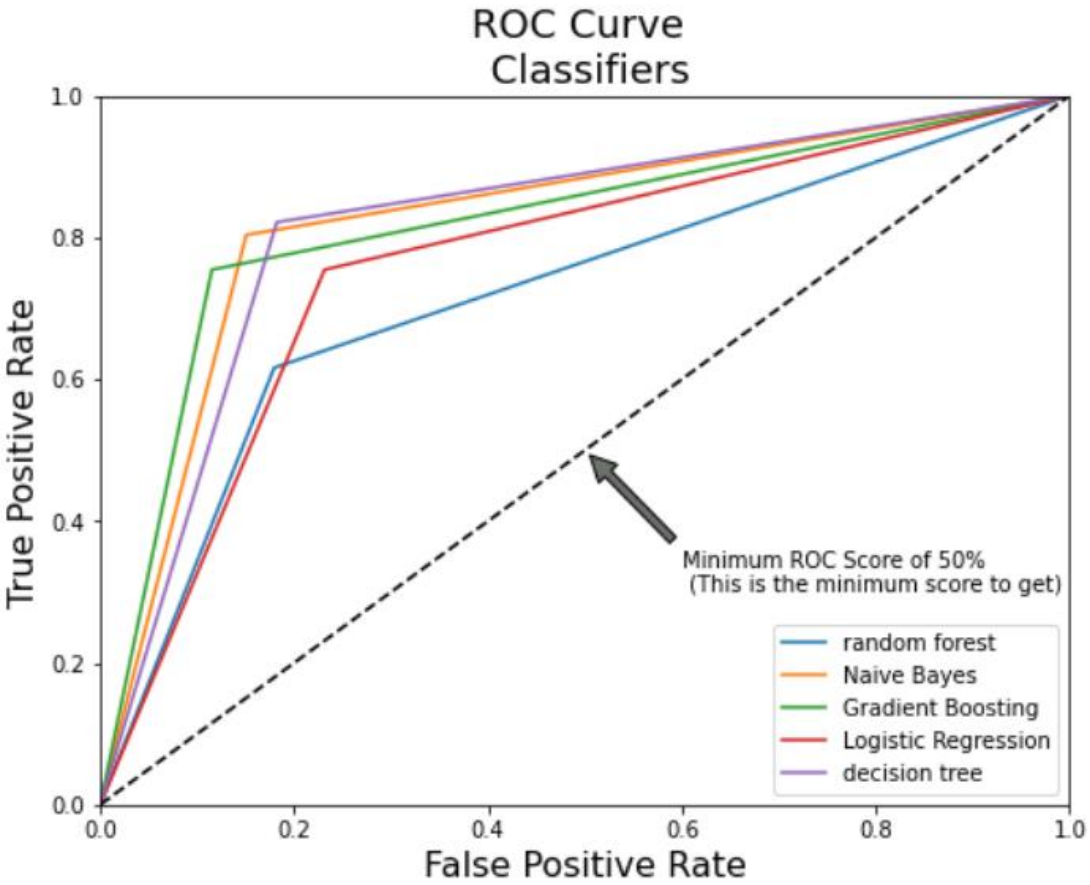


Figure 9: Modified TFIDF ROC Curve Classifiers

8. SUMMARY

This project report can be summarised through a brief explanation of the data cleaning, pre-processing and training of classifier models utilized. The data, a set of 31,000 tweets sampled directly from twitter is first carried through the cleaning process. At this time, we try to eliminate roadblocks that may hamper proper training of classifier models at the last step, so we try to clean data using techniques like removing @user tags, stemming, lemmatisation, removal of stop words etc. This data is then sampled and pre-processing algorithms: standard TFIDF and our modification to TFIDF algorithm is applied and word weights are calculated. Five standard classification models are then trained on this data and the results i.e. accuracy of each model, are plotted on a graph and are compared.

9. REFERENCES

[1] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[2] https://en.wikipedia.org/wiki/Gradient_boosting

[3] https://en.wikipedia.org/wiki/Linear_regression

[4] https://en.wikipedia.org/wiki/Decision_tree

[5] https://en.wikipedia.org/wiki/Random_forest

[6] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[7] https://en.wikipedia.org/wiki/Data_pre-processing