# Project - Data collection – Milestone 6

Madhumathy Kumaran
kumaran.m@emailaddress.edu

Percentage of Effort Contributed by Student :________97_______
Signature of Student :__Madhumathy Kumaran___________________________________
Submission Date:___ __________Apr 9 2023_____________________________

# Twitter Sentiment Analysis – Hate and Abusive speech Analysis

Madhumathy Kumaran

Master of science in Computer Science,

Northeastern University, Seattle

***Abstract*— Any communication that disparages a target group of people based on a trait like race, color, ethnicity, gender, sexual orientation, nationality, religion, or another feature is usually referred to as hate speech. The volume of hate speech is continuously rising as a result of social media's enormous growth in user-generated content. Along with the phenomenon's effects on society, interest in online hate speech identification and the automation of this activity, has risen steadily over the past few years. This study describes a dataset of hate speech that includes thousands of words that have been manually classified as containing or not hate speech.**

## 1. INTRODUCTION

In recent years, various studies looked at a variety of vulgar and unfriendly expressions for people based on various race, ethnicity issues. utilizing Twitter data between October 22 and October 28 and the Tweet Binder analytics application. Prior to Musk's acquisition, the seven-day average of Tweets employing the researched hate phrases was never more than 84 times per hour. However, during the hours of midnight to noon on October 28, 2022 (shortly after Musk's acquisition), 4,778 tweets containing the hate speech in question were sent out. "The notion of loosening social media moderation has always fueled the propagation of prejudice and conspiracies. This is especially risky for young individuals using platforms, according to Bond Benton, a professor at Montclair who worked on this study and studies online extremism. The outlying problem targets the platforms with less to no lax or moderation towards hate speech and abuse and the effect it has on the users. The dataset used in this analysis is [1]. The recent improvement in technology has altered the ways people convey and receive information. Regardless of the diversity in the behaviour and character, everyone wants a platform that can be a place to share the thoughts and voice out the issues. Due to the various hate speech that gets delivered in these public forums, it becomes a mental issue and hazard to various weak minded and loose hearted. The main aim of this study is to compare various methods to classify the hate speech from the twitter data.

## 2. Problem Statement

The problem we address in this paper can be described as follows: Given a set of twitter tweets written by several online users. Each text is labelled, and annotated, and various models are used. The following question is answered "which model can be used to automatically detect hate speech in twitter and classify them as abusive and non-abusive".

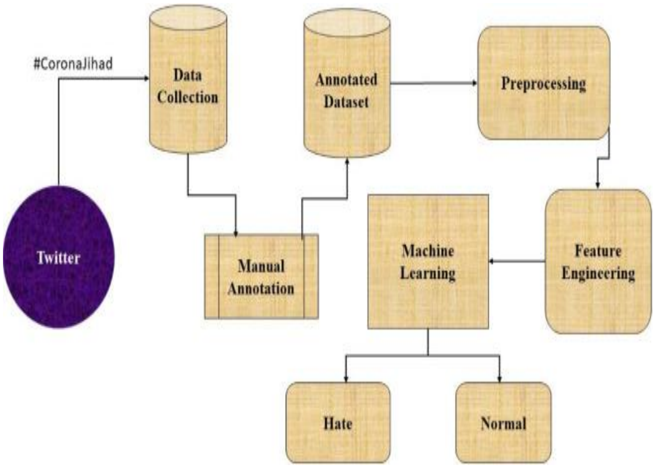
To answer this question, our main goals can be summarized as follows:
– To develop a novel method that can improve the state-of-art approaches within hate-speech classification, in terms of classification performance/accuracy.
– To investigate the impact of incorporating information about existing personalized labeled postings from users' history on the classification performance/accuracy.

We are doing a comparative study of various methods and concluding which is best and effective.

## 3. Proposed Approach

The proposed methodology which is being used for detecting hate speech using Machine Learning is shown in Fig depicts a series of steps:

(i) Data collection
(ii) Preprocessing
(iii) Feature Engineering
(iv) Machine Learning Classification
(v) Ensemble Learning Classification.



To begin with we are trying various models trained after N-grams and TFIDF and the results are compared using the various classifier models. To begin with we split the data set into train and test dataset and train them using various Baseline Models and proposed methods:
Baseline Models: To start with we start the modeling with Bog of words models as base,

1. Naïve Bayes model,
2. Logistic Regression
3. Random Forest classifier.

Neural Models: Some of the neural models this dataset can be compared are:

1. CNN
2. LSTM and Bi LSTM
3. Fast Text
4. Transformers

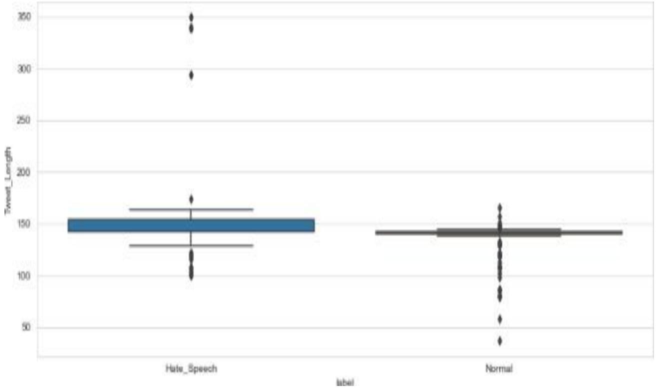
## 4. Data Used

**Dataset overview**: Our goal is to classify tweets into two categories, hate speech or non-hate speech. For this we have tweets from nearly 11k users online. Each tweets have mixed comments and hate tweets to help us understand the difference it has with the neutral comments.

The following data set is used,

**Data fields**: The data is stored as CSV and each data has five columns:

Count: number of users who tweeted(min is 3, sometimes more users tweeted
hate_speech = number of users who judged the tweet to be hate speech
offensive_language = number of users who judged tweet as offensive
neither = neither offensive nor hate speech
class = 0- hate speech 1 – offensive language 2 – neither tweet = comment made by the user.

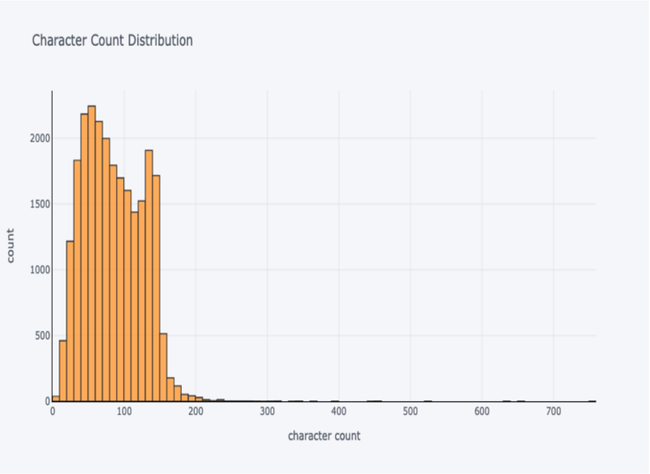


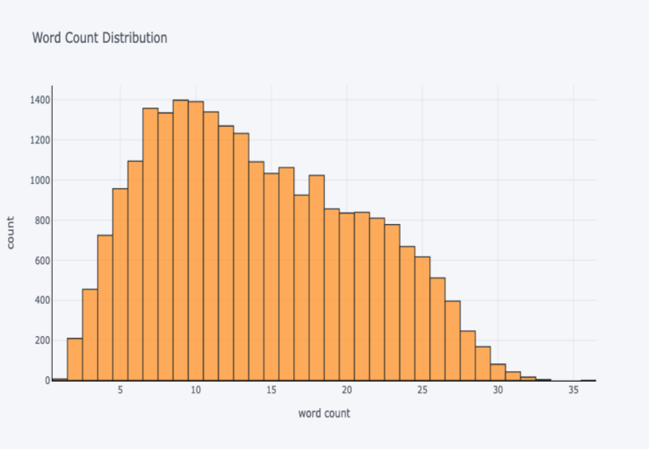
## 5. Explanatory Data Analysis

To begin our analysis, we start with some initial data analysis to get some better insight of the data used, we perform the following:

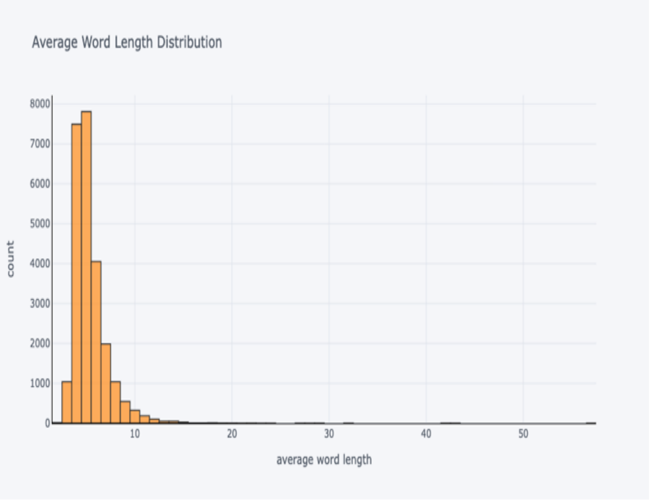
Explanatory data analysis is one of the most important parts of any machine learning workflow and its makes the patterns more easy to analyse.

1. **Word and character count of the data**: We find the number of words per tweet to understand the frequency of the data.



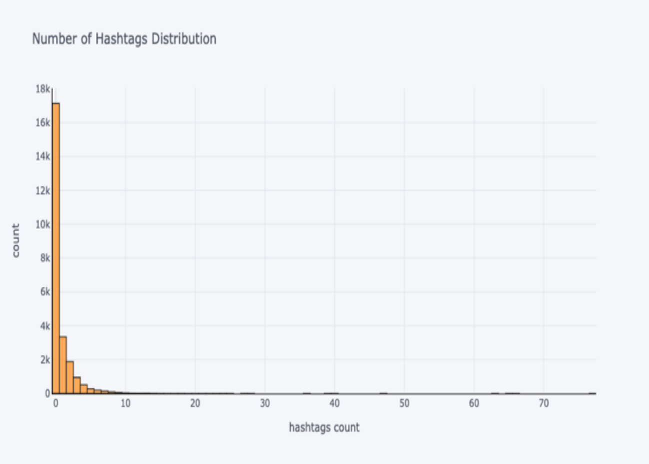


2. **Average word length per tweet**: We define the function to calculate the word length of tweet and average length of every tweet.

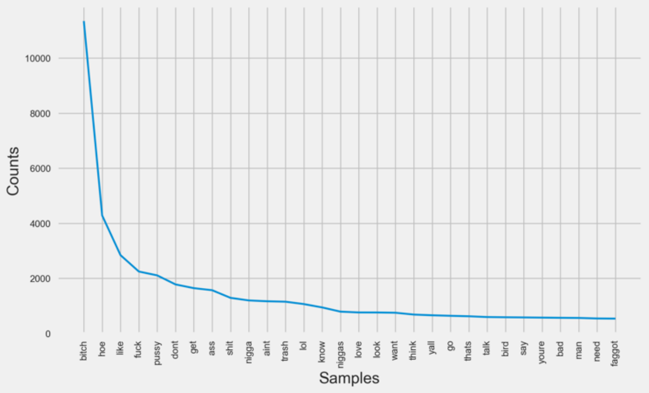


4. **Hashtag Distribution**:

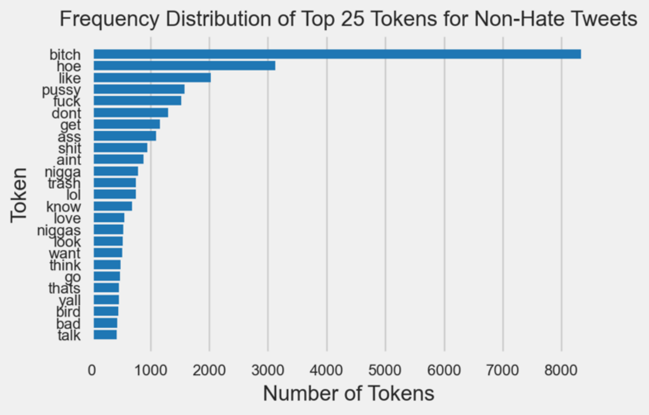
We create visualization for displaying hashtag distribution,



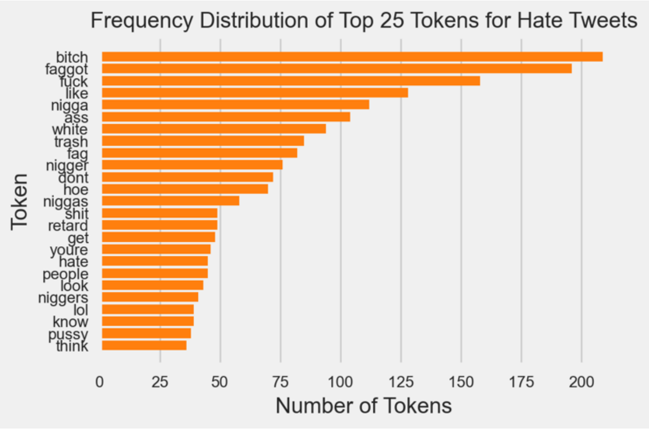
4. **Ploting words frequency distribution**:
Plotting words frequency distribution from the corpus,



5.**Frequency Distribution**:
Frequency distribution of top 25 tokens for non-hate tweets using yellow brick



Frequency distribution of top 25 tokens for non-hate tweets using yellow brick



## 6. Data Preprocessing and transformation

In the data preprocessing stage, the tasks involve removal of unnecessary columns from the datasets and enumerating the classes. For the third dataset, we retrieve the tweets corresponding to the tweet-ID present in the dataset. We convert the tweets to lowercase and remove the following unnecessary contents from the tweets:

- Space Pattern
- URLs
- Twitter Mentions
- Retweet Symbols
- Stopwords

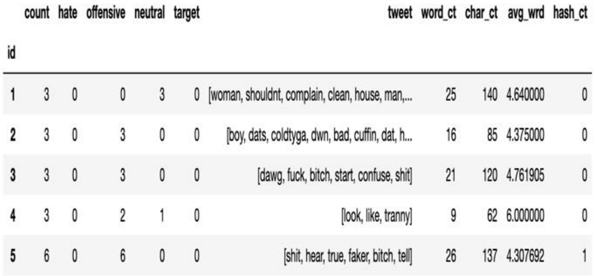
Here is the thought process involved with each of the specific steps we identified working with the dataset to prepare the data for the modeling process:

- We removed callouts or usernames, which is preceded by @. They contain no useful information.
- We removed character references, which includes HTML character references, but also emojis, unicode characters. We

decided not to convert any emojis into sentiment words.

- We removed the hash from the hashtags and decided to keep the hashtag text because they are often words or word-like and are used to connect similar ideas across the platform. We could analyze the hashtags in a future project.
- We removed the Twitter codes RT and QT for retweet and quotetweet. We decided to
- keep the retweeted words, as it conveys important information while others have removed all the text after RT.
- We removed the HTML links since a lot of users link a website reference as part of the tweet.
- We then removed any punctuation. We did not convert contractions into the uncontracted words.
- We then lowercased all the tweets for tokenizing.
- We removed any numbers and number containing words for tokenization and vectorizing.
- We removed any extra whitespace(s) between words and any leading and trailing whitespaces.

We use the Porter Stemmer algorithm to reduce the inflectional forms of the words. After combining the dataset in proper format, we randomly shuffle and split the dataset into two parts: train dataset containing 70% of the samples and test dataset containing 30% of the samples.



| | count | hate | offensive | neutral | target | tweet | word_ct | char_ct | avg_wrd | hash_ct |
|---|---|---|---|---|---|---|---|---|---|---|
| id | | | | | | | | | | |
| 1 | 3 | 0 | 0 | 3 | 0 | [woman, shouldnt, complain, clean, house, man,... | 25 | 140 | 4.640000 | 0 |
| 2 | 3 | 0 | 3 | 0 | 0 | [boy, dats, coldtyga, dwn, bad, cuffin, dat, h... | 16 | 85 | 4.375000 | 0 |
| 3 | 3 | 0 | 3 | 0 | 0 | [dawg, fuck, bitch, start, confuse, shit] | 21 | 120 | 4.761905 | 0 |
| 4 | 3 | 0 | 2 | 1 | 0 | [look, like, tranny] | 9 | 62 | 6.000000 | 0 |
| 5 | 6 | 0 | 6 | 0 | 0 | [shit, hear, true, faker, bitch, tell] | 26 | 137 | 4.307692 | 1 |

### C. Feature Extraction

We extract the n-gram features from the tweets and weight them according to their TFIDF values. The goal of using TFIDF is to reduce the effect of less informative tokens that appear very frequently in the data corpus. Experiments are performed on values of n ranging from one to three. Thus, we consider unigram, bigram and trigram features. The formula that is used to compute the TFIDF of term t present in document d is:

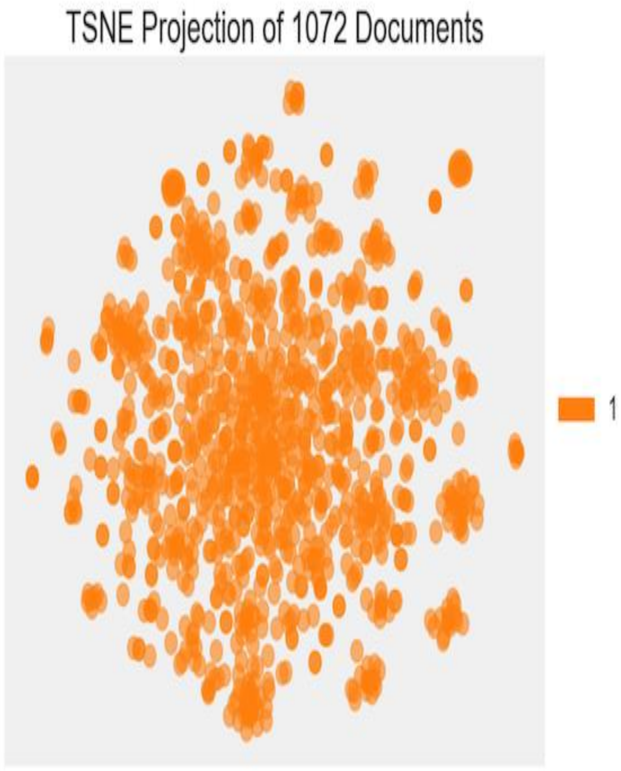
$$tf\ idf(d, t) = tf(t) * idf(d, t)$$

Also, both L1 and L2 (Euclidean) normalization of TFIDF is considered while performing experiments. L1 normalization is defined as:

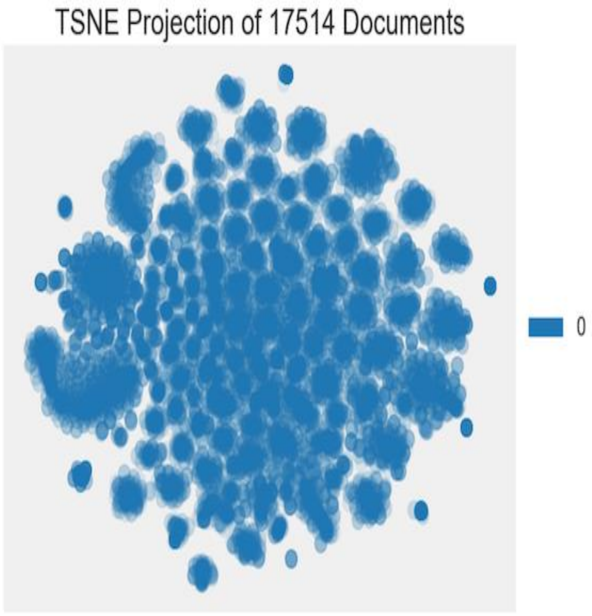
vnorm = v |v1| + |v2| + ... + |vn|
where n in the total number of documents.
Similarly, L2 normalization is defined as:
vnorm = v p v 2 1 + v 2 2 + ... + v 2 n
We feed these features to machine learning models.

This paper has a direct focus on classifying speech into abuse and non-abuse speech. This project has nearly 30000 tweets from the public repository. The data preprocessing involves two steps, Bag of words and Term Frequency Inverse Document Frequency (TFIDF). The **bag-of-words** approach is a simplified representation used in natural language processing and information retrieval. In this approach, a text such as a sentence or a document is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity [2]. **TFIDF** is a numerical statistic that is intended to reflect how important a word is to a document in a collection. It is used as a weighting factor in searches of information retrieval, text mining, and user modeling [2]. This approach can help in decreasing the hate and abuse comments or at least remove them from the public view giving a healthy environment to voice out the ideas and thoughts of the users.

TNSE Visualization for the positive class:

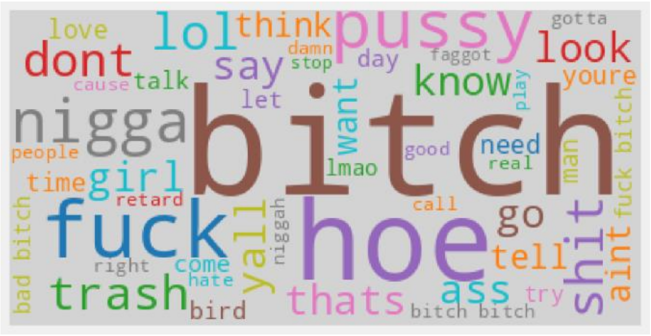


TNSE Visualization of Negative class:

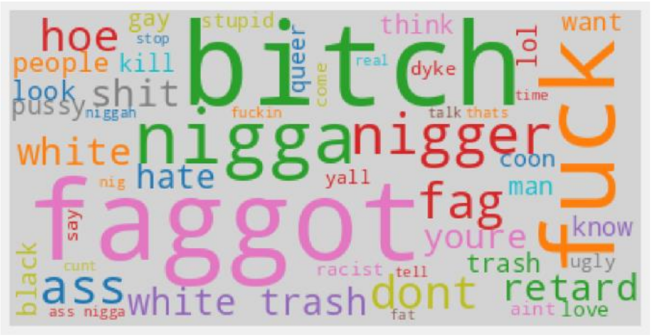


**Word cloud Visualisation**:

The tf-idf word embedding methods are visualised using the word cloud for both positive and negative tweets,



Similarly, we can visualise the word cloud diagram for positive tweets,



## 7. Modeling

This dataset is considered using major machine learning models like Naïve Bayes, Logistic Regression and Random Forest classifier and the results are compared. The accuracy of every model is compared using the confusion matrix and the

accuracy score is deduced using ruc score.

The data set is fit and trained into various training models and compared using the various classifiers, the following steps are followed,

1. Data set gets preprocessed
2. Data gets split into training and test data
3. Vectorizing is done through TF-IDF

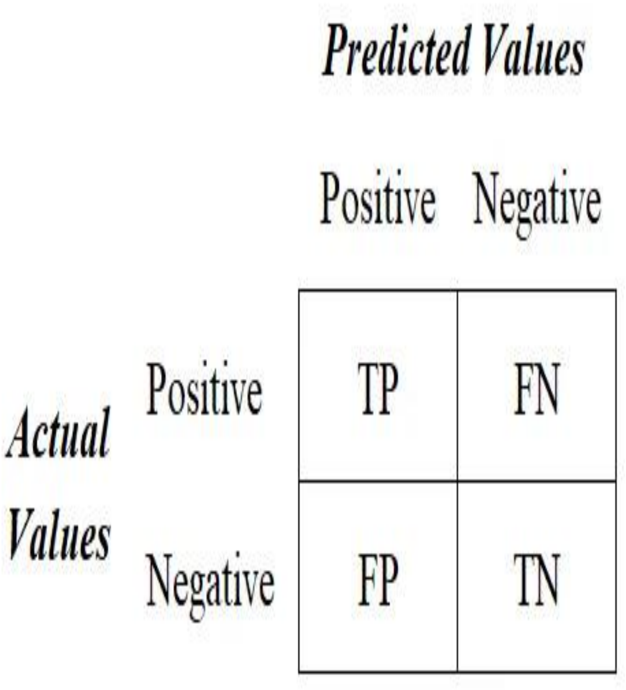
4. Data is fit into the model
5. Model gets trained
6. Metrics is used to compare the performance.

**Performance Evaluation**

The resulting classification models are evaluated to determine which performs best based on the degree of effectiveness. While good classification models are useful for prediction purposes, poor classification models lead to unreliable outcomes, and thus, are not useful for the user.

Performance evaluation metrics are based on the total number of the following variables in the confusion matrix:

- **True Positives**: outcome correctly predicted as positive class
- **True Negatives**: outcome correctly predicted as negative class
- **False Positives**: outcome incorrectly predicted as positive class
- **False Negatives**: outcome incorrectly predicted as negative class



| | | Predicted Values | |
|---|---|---|---|
| | | Positive | Negative |
| Actual Values | Positive | TP | FN |
| | Negative | FP | TN |

We have measured through the following scores,

- Accuracy
- F1 Score
- ROC-AUC
- Recall
- Precision
- PR-AUC
- Log loss

Let’s see some of the classifier models and their performance,

**Multi Naïve Bayes**

Multi Naive Bayes is a classification algorithm that is often used in natural language processing (NLP) tasks such as sentiment analysis, spam detection, and text classification. In the context of Twitter hate speech detection, multi–Naive Bayes can be used to classify tweets as either containing hate speech or not containing hate speech

The first step in using multi–Naive Bayes for Twitter hate speech detection is to gather a dataset of tweets that have been manually labeled as either containing hate speech or not containing hate speech.
The multi–Naive Bayes algorithm works by calculating the probability that a tweet belongs to a particular class (in this case, either hate speech or not hate speech) based on the words that appear in the tweet.

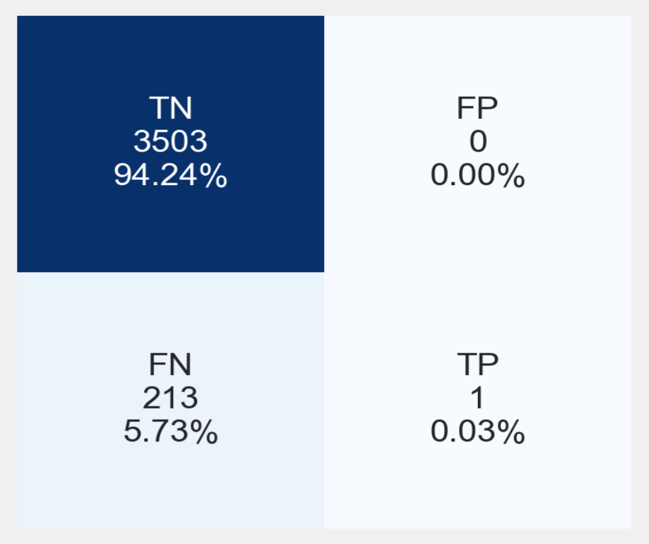
P(X | C) = P(x1 | C) * P(x2 | C) * …* P(xn | C)

Where:

- P(x1 | C) is the probability of the first feature or word given class C
- P(x2 | C) is the probability of the second feature or word given class C
- P(xn | C) is the probability of the nth feature or word given class C

Naïve Bayes classifier has been visualised as follows,

Accuracy: 0.9426957223567393
F1 Score: 0.009302325581395347
ROC-AUC: 0.7158910519954859
Recall: 0.004672897196261682
Precision: 1.0
PR-AUC: 0.17537404097663506
Log loss: 1.9792196319924609



| | |
|---|---|
| TN<br>3503<br>94.24% | FP<br>0<br>0.00% |
| FN<br>213<br>5.73% | TP<br>1<br>0.03% |

## Decision Tree Classifier

The first step in using a decision tree classifier for Twitter hate speech detection is to gather a dataset of tweets that have been manually labeled as either containing hate speech or not containing hate speech. Once this dataset has been collected, it can be split into a training set and a testing set. The training set is used to train the decision tree classifier, while the testing set is used to evaluate the performance of the classifier.

The decision tree algorithm works by recursively partitioning the training data into subsets based on the values of different attributes, or features, of the data. Each split in the tree represents a decision based on the value of a particular feature.

To use a decision tree classifier for Twitter, hate speech detection, the algorithm would first need to be trained on the labeled dataset. During training, the algorithm would learn the optimal splits based on the features of the training data. Once the algorithm has been trained, it can be used to classify new tweets as either containing hate speech or not containing hate speech based on the splits in the decision tree.

It is worth noting that decision tree classifiers can be prone to overfitting, where the model is too complex and fits the training data too closely, resulting in poor generalization to new data.

The formula for calculating the information gain of an attribute A is:

Information Gain(A) = Entropy(S) - Σ ((|Sv| / |S|) * Entropy(Sv))

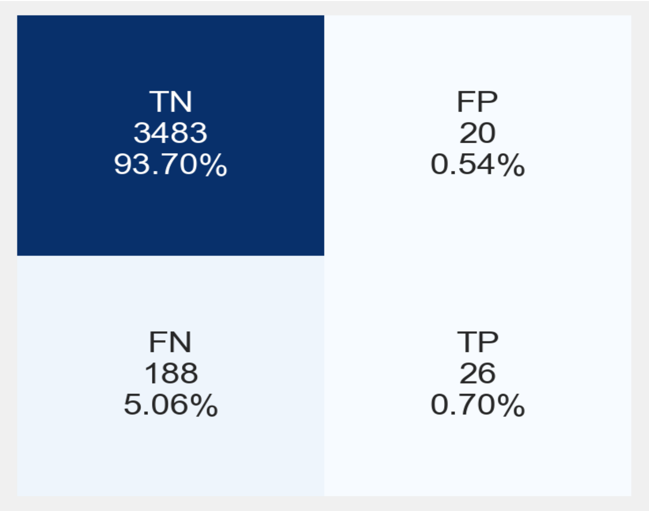
where:
S is the set of instances to be classified
Sv is the subset of instances in S for which attribute A has the value v
Entropy(S) is the entropy of set S, which measures the impurity of the set and is defined as:
Entropy(S) = -Σ (pi * log2(pi)), where pi is the probability of an instance in S belonging to class i
|Sv| is the number of instances in subset Sv
|S| is the total number of instances in set S
The formula for calculating the Gini impurity of an attribute A is:

Gini(A) = 1 - Σ (pi^2)

where pi is the probability of an instance in the set S

Decision tree classifier confusion matrix can be seen below fig,

Accuracy:  0.9227871939736346
F1 Score:  0.2878411910669975
ROC-AUC:  0.7158910519954859
Recall:  0.27102803738317754
Precision:  0.30687830687830686
PR-AUC:  0.17537404097663506
Log loss:  2.666864022761023



| | |
|---|---|
| TN<br>3483<br>93.70% | FP<br>20<br>0.54% |
| FN<br>188<br>5.06% | TP<br>26<br>0.70% |

**Random Forest Classifier**

Random forest is a type of ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the classification. In the context of Twitter hate speech detection, a random forest classifier can be used to classify tweets as either containing hate speech or not containing hate speech.

The first step in using a random forest classifier for Twitter hate speech detection is to gather a dataset of tweets that have been manually labeled as either containing hate speech or not containing hate speech. Once this dataset has been collected, it can be split into a training set and a testing set.

To use a random forest classifier for Twitter hate speech detection, the algorithm would first need to be trained on the labeled dataset. During training,
the algorithm would create multiple decision trees on different subsets of the training data and features. Once the decision trees have been trained, they can be combined to classify new tweets as either containing hate speech or not containing hate speech.

The formula for the random forest algorithm is:

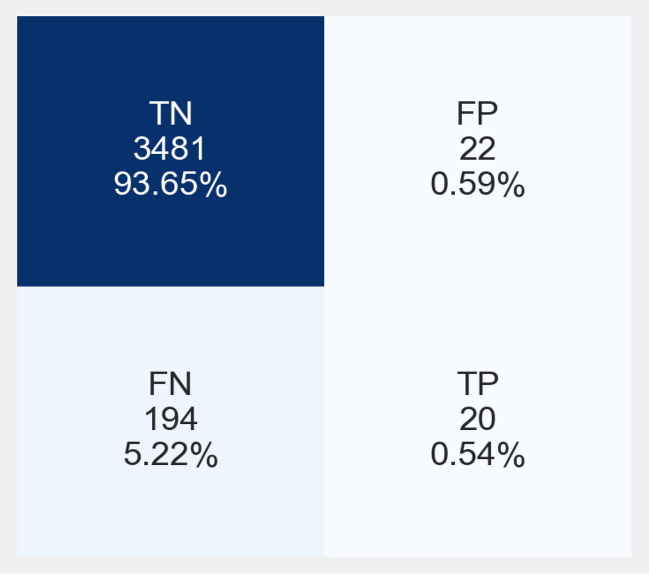
1. For each tree in the forest:
    - Randomly select a subset of the training data, with replacement
    - Randomly select a subset of the features to use for the tree
    - Build a decision tree using the selected subset of the data and features

2. To classify a new instance:
    - Pass the instance through each decision tree in the forest
    - Count the number of times the instance is classified as each class
    - The class with the most votes is the final prediction.

In addition to the above formula, there are a few other important considerations when using a random forest classifier, such as hyperparameter tuning and feature selection.

Random forest classifier has the metrics as follows,

Accuracy:  0.9418886198547215
F1 Score:  0.15625
ROC-AUC:  0.8270914649926232
Recall:  0.09345794392523364
Precision:  0.47619047619047616
PR-AUC:  0.3209943416834018
Log loss:  2.00710069745606



| | |
|---|---|
| TN<br>3481<br>93.65% | FP<br>22<br>0.59% |
| FN<br>194<br>5.22% | TP<br>20<br>0.54% |

## Logistic Regression

Logistic regression is a statistical modeling technique used for binary classification tasks, which can be applied to Twitter hate speech detection. The aim of logistic regression is to predict the probability of a binary outcome, such as whether a tweet contains hate speech or not, based on a set of input variables or features.

The formula for logistic regression can be represented as:

$p = 1 / (1 + e^{(-z)})$

where:

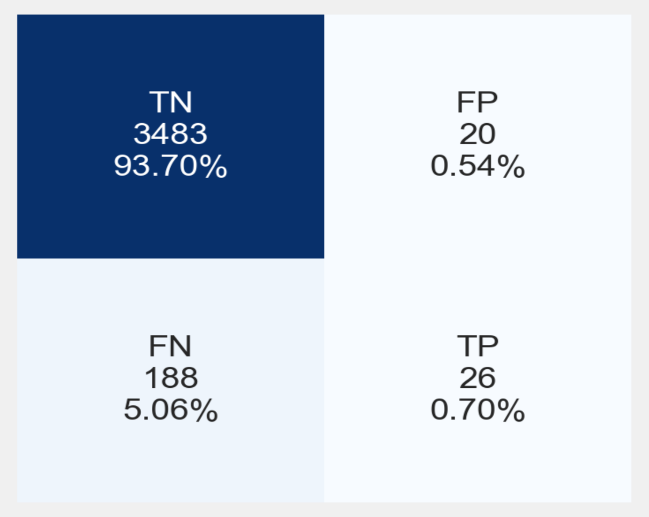
- p is the probability of the binary outcome,
- z is the input to the logistic function, and e is the base of the natural logarithm.

In the case of Twitter hate speech detection, we can represent each tweet as a set of features, such as the presence or absence of certain words or phrases, the use of punctuation or emojis, or the length of the tweet. These features can be used as input to the logistic regression model to predict the probability of the tweet containing hate speech.

To train the logistic regression model, we need a labeled dataset of tweets that have been manually classified as containing hate speech or not.

The logistic regression model can also be regularized to prevent overfitting by adding a penalty term to the cost function. Regularization techniques such as L1 regularization or L2 regularization can be used to shrink the coefficients of the input features towards zero, which helps to reduce the complexity of the model and improve its generalization performance.

Logistic regression has some different results with the below metrics,
Accuracy:  0.9440408931934355
F1 Score:  0.2
ROC-AUC:  0.8740065257816397
Recall:  0.12149532710280374
Precision:  0.5652173913043478
PR-AUC:  0.35577806219015606
Log loss:  1.9327633796314303



| | |
|---|---|
| TN<br>3483<br>93.70% | FP<br>20<br>0.54% |
| FN<br>188<br>5.06% | TP<br>26<br>0.70% |

**Gradient Boosting Classifier**

Gradient boosting is an ensemble machine learning technique that combines multiple weak models to create a strong predictive model. Gradient boosting classifier is a type of gradient boosting algorithm that is specifically used for classification tasks, such as Twitter hate speech detection.

The formula for gradient boosting classifier can be broken down into the following steps:

- Initialize the model with a simple weak learner, such as a decision tree.
- Train the weak learner on the training data and calculate the residuals, which are the differences between the predicted and actual labels.
- Train a new weak learner on the residuals from the previous step and add the new learner to the model by combining it with the previous learner.

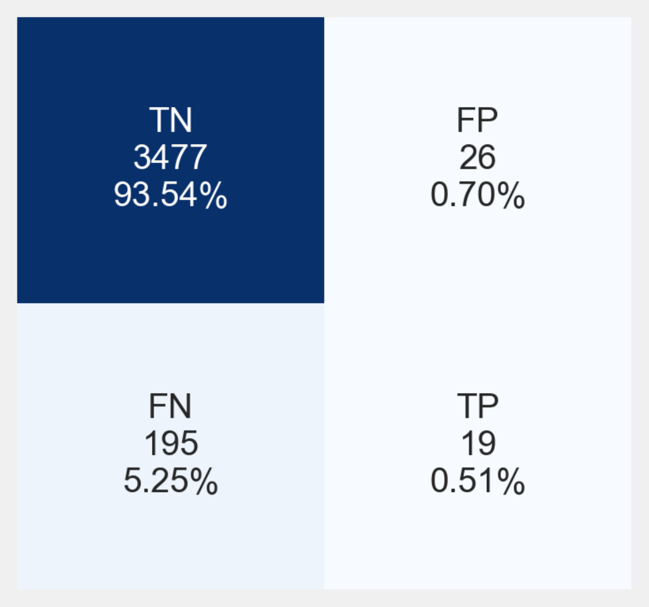
Repeat steps 2-3 until the desired number of weak learners has been trained, or until
To make a prediction for a new instance, pass it through each of the trained weak learners and combine their predictions using a weighted average or a voting scheme.

To make a prediction for a new instance: Pass the instance through each of the weak learners in the model Combine the predictions using a weighted average or a voting scheme. In the case of Twitter hate speech detection, we can represent each tweet as a set of features, such as the presence or absence of certain w
ords or phrases, the use of punctuation or emojis, or the length of the tweet. These features can be used as input to the gradient boosting classifier model to predict the probability of the tweet containing hate speech.

To train the gradient boosting classifier model, we need a labeled dataset of tweets that have been manually classified as containing hate speech or not. The dataset is split into a training set and a testing set, and the model is trained on the training set using an optimization algorithm such as gradient descent. The performance of the model is then evaluated on the testing set using metrics such as accuracy, precision, recall, and F1-score.

Below are the statistics of the gradient boosting,

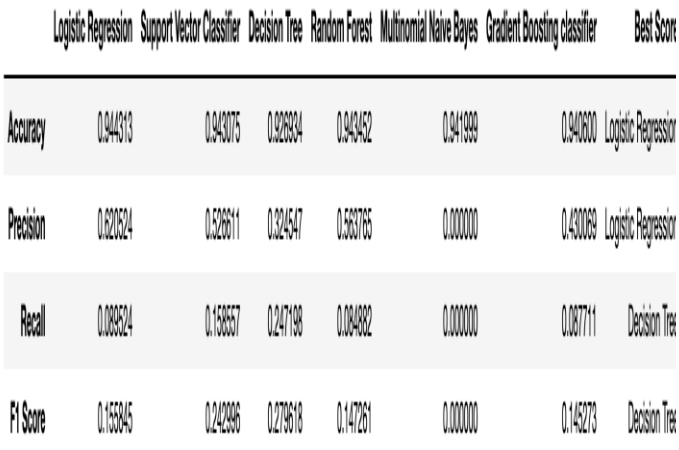
Accuracy:  0.9405434490180253
F1 Score:  0.1467181467181467
ROC-AUC:  0.8394506977997499
Recall:  0.08878504672897196
Precision:  0.4222222222222222
PR-AUC:  0.3344834312611418
Log loss:  2.0535621126737817



| | |
|---|---|
| TN<br>3477<br>93.54% | FP<br>26<br>0.70% |
| FN<br>195<br>5.25% | TP<br>19<br>0.51% |

## 7. Performance Evaluation

We have visualised the various classifiers to compare the performance using the scores of various metrics:

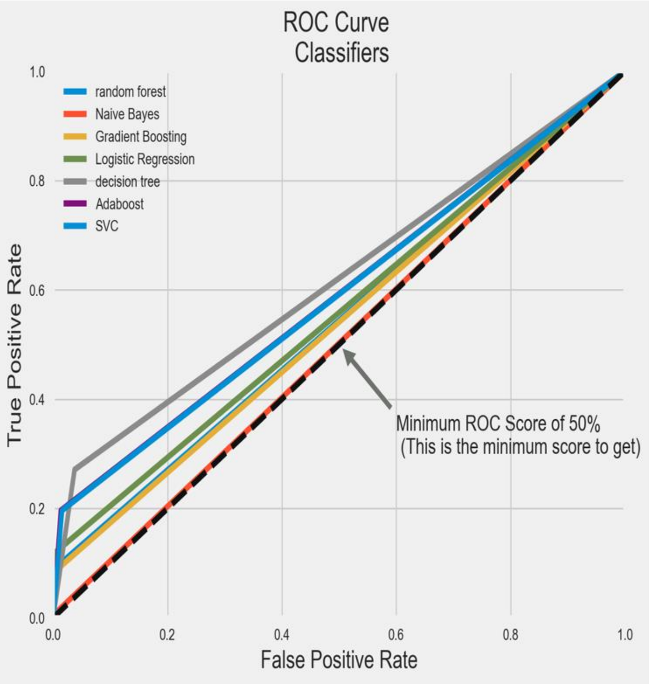


| | Logistic Regression | Support Vector Classifier | Decision Tree | Random Forest | Multinomial Naive Bayes | Gradient Boosting classifier | Best Scor |
|---|---|---|---|---|---|---|---|
| Accuracy | 0.944313 | 0.943075 | 0.926934 | 0.943452 | 0.941999 | 0.940600 | Logistic Regressio |
| Precision | 0.620524 | 0.526611 | 0.324547 | 0.563765 | 0.000000 | 0.430069 | Logistic Regressio |
| Recall | 0.089524 | 0.158557 | 0.247198 | 0.084882 | 0.000000 | 0.087711 | Decision Tre |
| F1 Score | 0.155845 | 0.242996 | 0.279618 | 0.147261 | 0.000000 | 0.145273 | Decision Tre |

Fig:  Comparison various classifier models

**Metrics using ROC – Curve and Roc- curve:**

Logistic Regression Score:  0.5578929675765232
Naive Bayes Score:  0.5023364485981309
Random Forest Score:  0.5435888063902503
Decision Tree Score:  0.6168157600561335
Grad Boosting Score:  0.5406814185971437
Ada boost Score:  0.5914223055805303
SVC Score:  0.5904231619893229



## 8. References

[1] https://www.sciencedirect.com/science/article/pii/S2667096822000635?via%3Dihub#bbib0009

[2] https://folk.idi.ntnu.no/heri/papers/pitsilis_APIN2018.pdf

[3] https://aclanthology.org/W16-5618.pdf

[4] https://towardsdatascience.com/detecting-hate-tweets-twitter-sentiment-analysis-780d8a82d4f6

[5] https://towardsdatascience.com/machine-learning-classifiers-comparison-with-python-33149aecdbca