

# **HATE SPEECH DETECTION USING TWITTER**

A project dissertation submitted to Bharathidasan University  
in partial fulfillment of the requirements  
for the award of the Degree of

## **MASTER OF SCIENCE IN DATA SCIENCE**

*Submitted by*

**MAHALAKSHMI.S (205229118)**

*Guided by*

**Dr. K. RAJKUMAR M.Sc., M.Phil., Ph.D.,  
Head, Department of Data Science**



### **DEPARTMENT OF DATA SCIENCE BISHOP HEBER COLLEGE (AUTONOMOUS)**

(Nationally Reaccredited at the 'A' Grade by NAAC with the CGPA of 3.58 out of 4)  
(Recognized by UGC as "College with Potential for Excellence")  
(Affiliated to Bharathidasan University)

**TIRUCHIRAPPALLI 620017**

**MAY 2022**

## DECLARATION

I hereby declare that the project work presented is originally done by me under the guidance of Dr. K. Rajkumar, M.Sc., M.Phil., P.hD., Department of Data Science, Bishop Heber College (Autonomous), Tiruchirappalli 620017, and has not been included in any other thesis/project submitted for any other degree.

Name of the Candidate : MAHALAKSHMI.S

Register Number : 205229118

Batch : 2020-2022

*S. Mahalakshmi*  
Signature of the Candidate



DEPARTMENT OF DATA SCIENCE  
BISHOP HEBER COLLEGE (AUTONOMOUS)

(Nationally Reaccredited at the 'A' Grade by NAAC with the CGPA of 3.58 out of 4)

(Recognized by UGC as "College with Potential for Excellence")

(Affiliated to Bharathidasan University)

TIRUCHIRAPPALLI 620017

Date: 28/5/22

Course Title: Project

Course Code: P19DS4PJ

**BONAFIDE CERTIFICATE**

This is to certify that the project work titled "HATE SPEECH DETECTION USING TWITTER" is a bonafide record of the project work done by Mahalakshmi S, 205229118, in partial fulfillment of the requirements for the award of the degree of MASTER OF SCIENCE IN DATA SCIENCE during the period 2020 - 2022.

The Viva-Voce examination for the candidate Mahalakshmi S, 205229118, was held on 31/5/22.

Signature of the HOD

Signature of the Guide

Examiners:

1. J. Abdulaziz 31/5

2.

## ACKNOWLEDGEMENTS

I sincerely thank **Dr. D. PAUL DHAYABARAN, M.Sc., M.Phil., PGDCA., Ph.D., Principal,** Bishop Heber College, Trichy, for providing me with sufficient facilities which contributed to the successful completion of the project.

I wish to place on record my gratitude to **Dr. K. RAJKUMAR, M.Sc., M.Phil., Ph.D., Associate Professor and Head, Department of Data Science (S.F),** for his motivation and guidance through the course of my project work.

I am blessed to have **Dr. K. RAJKUMAR, M.Sc., M.Phil., Ph.D., Associate Professor and Head, Department of Data Science** Bishop Heber College, as my research adviser. I wholeheartedly thank him because this is possible only because of his patience support and ready guidance. I wish to thank you for being kind and considerate in all respects towards me right from the beginning. His untiring help during my difficult moments, his motivation and his kindness helped me and in for completing my project work successfully.

I am grateful to my parents for their blessing, love and sacrifice in which my life was built. I also thank my friend, especially for their help and support in shaping the project.

**MAHALAKSHMI S**


## Document Information

---

Analyzed document	205229118_Mahalakshmi_HATE SPEECH DETECTION USING TWITTER .pdf (D138168805)
Submitted	2022-05-27T08:04:00.0000000
Submitted by	Librarian
Submitter email	librarypds@bhc.edu.in
Similarity	1%
Analysis address	librarypds.bhc@analysis.orkund.com

## Sources included in the report

---

<b>SA</b>	<b>Updated_Research_Paper.docx</b> Document Updated_Research_Paper.docx (D110508504)	 <b>1</b>
-----------	---	--

---

# ABSTRACT

Social media as well as other online platforms are playing a wide role in the breeding and spread of hateful content eventually leading to hate crime. The use of social media has been increasing day by day and also hate Speech is also increasing along with the number of users. As a result, organizations face a difficult task in monitoring each and every user's tweet, thus we're working on a machine learning model to detect hate speech tweets automatically which saves a lot of resources for companies. In this problem, the data set given has only 25% of hate tweets, so when we make a model using this data the model performs better on the normal tweets which do not break our problem. So, we must oversample the hate tweets in order for the two classes to be equal and our model to work properly. Hence, the aim of this project is to analyze the comments on social networks using NLP (Natural Language Processing) techniques, Machine learning algorithm, and deep learning algorithm.

The objective of this project is to combat online harassment and aggression by creating a prototype that can automatically detect cyberbullying and abusive behavior on social media and online forums by extracting, collecting, labeling data collection, and improving accuracy by preprocessing, cleaning, and experimenting with different characteristics into one of the various categories and analysis and evaluation of the best model.

The preprocessing step involved here is dropna, stop words, lemmatization, removing all the special characters, converting all letters to lower case and joining all words back to the text the and word cloud method for visualization. In order to increase the accuracy of the model, the next step would be feature engineering – Count Vectorizer and Tf-idf transformer is a great tool provided by the sci-kit learn library in Python. For evaluation and analysis of the result, various accuracy metrics – model accuracy score, precision, recall, and F1score will be noted. Then analyze or predict the tweets whether not hate tweets are labeled as '0' or hate tweets labeled as '1'.

As a result, the goal of this project is to analyze tweets on Twitter using Natural Language Processing (NLP), ML algorithms and DL algorithms. The results of classifying the gathered training dataset and test dataset, and also various metrics of evaluation of the performance used after training the dataset and validating it with a test dataset. Training accuracy varied from 70% to 90% for the ML classifier while the test accuracy lies between 30% to 45%

# TABLE OF CONTENTS

Chapter	Title	Page No
	<b>Abstract</b>	vi
	<b>List of Figures</b>	viii
	<b>List of Tables</b>	ix
<b>1</b>	<b>Introduction</b>	01
1.1	Motivation	
1.2	Existing Systems/Products and Solutions	
1.3	Product Needs and Proposed System	
1.4	Product Development Timeline	
<b>2</b>	<b>Related Work and Solutions Review</b>	05
2.1	Automatic Hate Speech Detection Using Twitter	
2.2	Hate Speech Detection on Twitter	
2.3	Detecting Insulting Comments on Twitter	
<b>3</b>	<b>Data Collection</b>	08
3.1	Description of the Data	
3.2	Source and Methods of Collecting Data	
<b>4</b>	<b>Preprocessing and Feature Selection</b>	09
4.1	Overview of Preprocessing Methods	
4.2	Overview of Feature Selection Methods	
4.3	Preprocessing and Feature Selection Steps	
<b>5</b>	<b>Model/Product Development</b>	10
5.1	Model Architecture	
5.2	Algorithms Applied	
5.3	Training Overview	
<b>6</b>	<b>Experimental Design and Evaluation</b>	18
6.1	Experimental Design	
6.2	Experimental Evaluation	
6.3	Customer Evaluation and Feedback	
<b>7</b>	<b>Model/Product Optimization</b>	27
7.1	Overview of Model Tuning and Best Parameter Selection	
7.2	Model Tuning Process and Experiments	
<b>8</b>	<b>Product Delivery and Deployment</b>	30
8.1	User Manuals	
8.2	Delivery Schedule	
8.3	Deployment Process	
<b>9</b>	<b>Conclusion</b>	33
9.1	Summary	
9.2	Limitation and Future Work	
	<b>References</b>	34
	Appendix-A: Data Set	36
	Appendix-B: Source Code	37
	Appendix-C: Output Screenshots	50

## LIST OF FIGURES

FIG NO	DESCRIPTION	PAGE NO
1.4	Product Development Timeline	04
5.1	Model Architecture	10
5.1.1	MLP Classifier	11
6.1.1	CV with Accuracy Score	20
6.1.2	CV with F1 Score	20
6.1.3	CV with Precision Score	21
6.1.4	CV with Recall Score	21
6.1.5	Tf-Idf with Accuracy Score	22
6.1.6	Tf-Idf with F1 Score	22
6.1.7	Tf-Idf with Precision Score	23
6.1.8	Tf-Idf with Recall Score	23
6.1.9	ROC Curve Classifiers of CV	24
6.1.10	ROC Curve Classifiers of Tf-Idf	24



## LIST OF TABLES

TABLE NO	DESCRIPTION	PAGE NO
1.1	Existing System	03
6.2.1	Experimental-1	25
6.2.2	Experimental-2	25
6.2.3	Experimental-3	26
8.2	Delivery Schedule	31

# Chapter 1

## INTRODUCTION

### 1.1 Motivation

Social media is a fashionable and, above all, simple means for people to publicly share their ideas and opinions while also interacting with others online. It is a stage in which people are easily harassed or abused by others, who express hate in various forms such as sexism, racism, politics, and so on. The motivation for the work is to learn the application and implementation of Natural Language Processing techniques with Machine Learning and Deep Learning algorithms in a real-world problem.

### 1.2 Existing Systems and Solutions

The purpose of this paper is to discover the best technique for classifying such remarks by examining and experimenting with various strategies. NLP and machine learning algorithms are used to analyze social remarks and identify the aggressive effect of an individual or a group. The results of classifying the obtained training and test datasets were offered by Impermium on Kaggle. Parts of speech, the Bag of Words, and lexical features are all useful in this context for their investigation. Natural Language Processing and Machine Learning techniques are used to analyze comments and predict aggressive behavior. The pipeline includes the extraction of appropriate data preprocessing, feature engineering, and classification. The main goal of this supervised learning task is to classify text from an online user into two categories: "Bully" and "Non-Bully" such as comments and status/post modifications. Machine learning models are built using the most basic Logistic Regression, Support Vector Machine, and the two most frequent ensemble approaches, Random Forest Classifier and Gradient Boosting Machine.

### Logistic Regression

Logistic regression is a supervised machine learning method that is similar to linear regression, but instead of utilizing a linear equation, it employs a sigmoid function to provide an output value in the range  $[0,1]$ , which can also be used for text classification. Various

hyperparameters are evaluated and modified to improve learning efficiency. For example, "C," a logistic regression parameter, is the inverse of regularisation strength.

## **SVM**

SVM translates lower-dimensional input into higher-dimensional data using nonlinear or linear mapping. It looks for the linear optimal dividing hyperplane within this new dimension to separate the tuples between the sets. Information from two array scans is always discriminated by a hyperplane for sufficient nonlinear scaling to a sufficiently high dimension. Various hyperparameters are evaluated and modified to improve learning efficiency. For example, the support vector machine parameter "C" is the inverse of regularisation strength. SVM values that are smaller indicate that the regularisation is stronger.

## **Random Forest Classifier**

The term "forest" refers to a grouping of trees. Random forest, on the other hand, is a collection of decision trees that is named random because it is a collection of largely uncorrelated trees that operate as a single model. The primary premise of random forest is that each tree expresses its forecast, and the random forest predicts its choice based on the majority decision. So, in our application for detecting hate speech, the trees label the statement as hate speech or not, and the random forest predicts whether the message is hate speech or not based on the majority choice. In a random forest, one of the parameters is the "number of trees in the forest".

## **Gradient Boosting Machine**

Gradient Boosting Machine (for Regression and Classification) is an ensemble method for forwarding learning. The guiding principle is that with increasingly finer approximations, good forecasting outcomes can be produced. Gradient boosting classifiers is a collection of machine learning techniques that combine numerous weaker models into a powerful, highly predictive output. Models of this type are popular because of their ability to accurately classify datasets. In most cases, decision trees are used to generate models for gradient boosting classifiers.

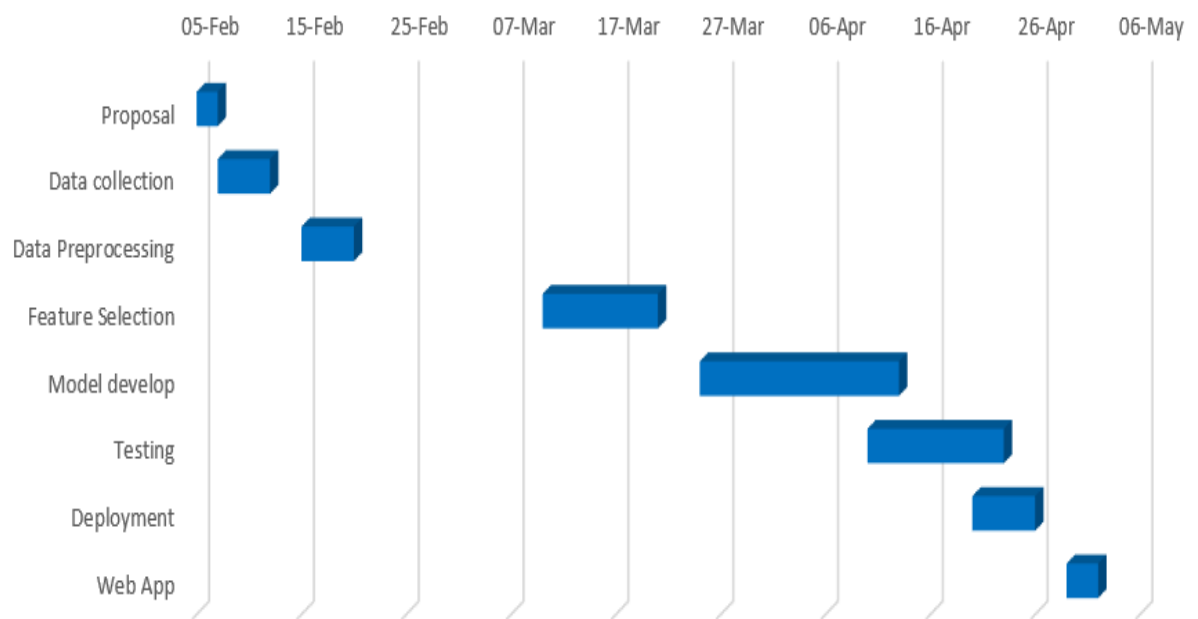
**Table 1.2 Existing Systems**

<b>Algorithms with TFIDF</b>	<b>Accuracy</b>
Logistic Regression	<b>0.90</b>
SVM	0.76
Random Forest	<b>0.90</b>
Gradient Boost	0.77

### **1.3 Product Needs and Proposed System**

Online Social Networks (OSNs) have become one of the most popular interactive platforms for communicating, exchanging, and spreading a large quantity of data about people's lives. Following data collection and extraction, the data set must be preprocessed or cleaned, which includes lowercasing, lemmatization, stop words removal, removal of URLs and punctuations and handling of missing values. The extraction and selection of a set of characterizing and discriminating features are the focus of most efforts in developing NLP techniques are count vectorizer, tf-idf and ML algorithms are XGBoost, Multinomial NB, AdaBoost and DL algorithm are MLP. In the proposed system, ML and DL algorithms for better accuracy value. By improving the model the accuracy value will be increased. Then compare all the models and choose the best accuracy model. In my project, **MLP Classifier** gives the best accuracy score (0.98%) than other models. Then analyzes or predicts the tweets whether not hate tweets are labeled as '0' or hate tweets are labeled as '1'.

## 1.4 Product Development Timeline



**Fig 1.4 Product Development Timeline**

## Chapter 2

### RELATED WORK AND SOLUTIONS REVIEW

#### 2.1 Automatic Hate Speech Detection Using Twitter

The Twitter data is analyzed using the KNLPEdNN-based hate speech detection algorithm. Initially, tweets are collected from the storm front and crowd flower datasets. An NLP technique is used to process the obtained data. The NLP tokenization process removes data characters, hashtags, user information, and other undesirable features. Using an optimal function, the features are analyzed using an ensemble deep learning classifier to predict whether replying Tweets will be classified as hate speech, offensive speech, or neither. and weight updating process. The approach is demonstrated using the Stormfront and Crowd-Flower Twitter datasets. It is demonstrated that the proposed automatic system ensures minimum loss function (MSE0.019, CEL-0.015 and LL-0.0238) and maximum prediction accuracy (98.71%) [1]. Using n-gram features weighted with TFIDF values, machine learning was used to detect hate speech and inflammatory language on Twitter. We performed a comparative analysis of Logistic Regression, Naive Bayes and Support Vector Machines on various sets of feature values and model hyperparameters. For the L2 normalization of TFIDF, the findings showed that Logistic Regression performs better with the ideal Ngram range of 1 to 3. Upon evaluating the model on test data, achieved 95.6% accuracy [2]. This paper uses Twitter hate speech and offensive identification dataset that is classified using the Random Forest method which will be compared with the results of its accuracy with AdaBoost and Neural Network to detect hate speech and crude speech. The detection results of hate speech and crude speech identification resulted in an accuracy of 0.722 for the Random Forest method and 0.708 using AdaBoost and 0.596 using the Neural Network method [3].

#### 2.2 Hate Speech Detection on Twitter

This paper used to test which model would perform better, researchers used two different types of data to detect hate speech in machine learning and deep learning using RNN. To improve accuracy in multiple methods of machine learning and deep learning, we try to extract hate speech from tweets to find the best method. And also extracted some data from the raw data at the ratio

of 10%-90% as test data and found that the overall performance of LR is better than SVM. In the case of using SVM, the performance of TF-IDF is more prominent. From the latest experimental progress, it can be concluded that using BiRNN can get a better result now to detecting hate speech from tweets. We will try to experiment with the other models [4]. we have used tf-idf and bag of words methods to extract features from the tweets. To classify hate speech from the tweets we have implemented machine learning algorithms like SVM, Logistic Regression and Random Forest. We can conclude from the results obtained that by using Data without preprocessing and machine learning models with default parameters, Random Forest with a bag of words gives the best performance with a 0.6580 F1 Score and 0.9629 Accuracy Score. After preprocessing and grid search SVM with Tf-IDF gives best the performance with a 0.7488 F1 Score and 0.9668 Accuracy Score [5]. Using a combination of lexicon-based and machine learning methodologies, this study used an emotive approach through sentiment analysis to predict hate speech in a text. The precision and recall results on predicting hate speech were always better than Davidson's, with the Random Forest method having the best precision, as shown in Table V, where PR and RC stand for "Precision" and "Recall," respectively. Davidson did not provide this information for comparison, but the overall results for each algorithm can be considered successful once the result for the Naive Bayes algorithm was obtained, with an average of 71.33 percent, and the best result was obtained for the SVM, with an average of 80.56 percent [6].

## 2.3 Detecting Insulting Comments on Twitter

In this paper, the results of classifying the gathered training dataset and test dataset provided by Imperium on Kaggle. Some possible features could be – Lexical Syntactic Features, TF-IDF (Term Frequency – Inverse Document Frequency), count of offensive words in a sentence, count of positive words in a sentence, count of second-person pronoun in a sentence, Character 4-gram and 5-gram count, Word/Document Vectors. Requirement of different kinds of features, for example, Latent Dirichlet allocation (LDA), Latent Semantic Analysis (LSA), Predictive Word embeddings like Word2Vec features and Doc2Vec features, etc. The experiments and overall work is that out of the method that was have evaluated, Logistic Regression and Random Forest Classifier trained on the feature stack performed better than Support Vector Machine and Gradient Boosting Machine in this particular case [7]. In this paper, a Stacked Weighted Ensemble (SWE) model is proposed for the detection of hate speeches. The model ensembles five standalone classifiers: Linear Regression, Naïve Bayes', Random Forest, Hard Voting and Soft Voting. The experimental results on a Twitter dataset have shown an accuracy of 95.54% in the binary classification of tweets into hateful speech and an improved performance is noted compared to the

standalone classifiers. SWE provides a higher accuracy rate of 95.54%, precision rate of 89%, recall rate of 74% and F-Measure rate of 77% [8]. GRU is one of the deep learning methods that have the ability to learn information relations from the previous time to the present time. In this research feature extraction used is word2vec, because it has the ability to learn semantics between words. In this research, the Gated Recurrent Unit (GRU) performance is compared with other supervised methods such as Support Vector Machine, Naive Bayes, Random Forest, and Logistic Regression. The result of experiments shows that the combination of word2vec and GRU gives the best accuracy of 92.96%. However, the use of word2vec in the comparison methods results in a lower accuracy than the use TF and TF-IDF features [9].



## Chapter 3

### DATA COLLECTION

#### 3.1 Description of the Data

The objective of this dataset is to use hate and not hate tweets. The dataset was gathered from the Twitter online social media network. It's publicly accessible on GitHub or Kaggle and other websites also. The dataset labeled as not hate tweets as '0' and hate tweets as '1'. 20 % of data is used as testing data and 80% of training data is from each class.

##### Column Names:

Id: the id column is a unique identifier of tweets.

Label: the label column is the labeling of the tweets as 0 and 1.

Tweet: the tweet column is a tweet get from Twitter.

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

#### 3.2 Source and Methods of Collecting Data

- The data is stored in an Excel spreadsheet as a CSV file.
- There are two columns in the data set labeled Hate Tweets as 1 and Not a Hate Tweets as 0.
- They are responsible for a Total of 29720 Not a Hate Tweets and a Total of 2242 Hate Tweets in the Dataset.

The link to the datasets is HTTPS: <https://github.com/NakulLakhotia/Hate-Speech-Detection-in-Social-Media-using-Python>

## Chapter 4

### PREPROCESSING AND FEATURE SELECTION

#### 4.1 Overview of Preprocessing Methods

The dropna method is used to eliminate the rows with null values as part of the preprocessing procedure. Stop words are English words that don't add much meaning to a sentence, lemmatization is the process of converting a word to its base form, removing all special characters, converting all letters to lower case and joining all words back to the text.

#### 4.2 Overview of Feature Selection Methods

The Count Vectorizer feature extraction method counts the number of times each word appears in a document to convert it into a vector. This is also known as Bag of Words (BoW), which implies counting the words and placing them in the bag regardless of their order or structure in layman's terms. A count matrix is transformed into a normalized tf or Tf-Idf representation using the Tf-Idf transformer. Tf stands for term frequency, while Tf-df stands for term frequency multiplied by the inverse document frequency. This is a popular term for weighting systems in information retrieval, and it's also useful in document classification.

#### 4.3 Preprocessing and Feature Selection Steps

The steps of the Preprocessing are :

- Remove stop words
- Analyze lemmatization words
- Remove special characters.
- Convert all letters to lower case
- Join all words back to text and word cloud method.

The steps of the Feature selection are as follows:

- Count vectorization
- Tf-Idf transformer

## Chapter 5

### MODEL DEVELOPMENT

In this project, we can implement Tf-idf Transformer and Count Vectorizer to extract the features, ML algorithms are XGBoost, MultinomialNB, AdaBoost and DL algorithm are MLP to classify the tweets.

#### 5.1 Model Architecture

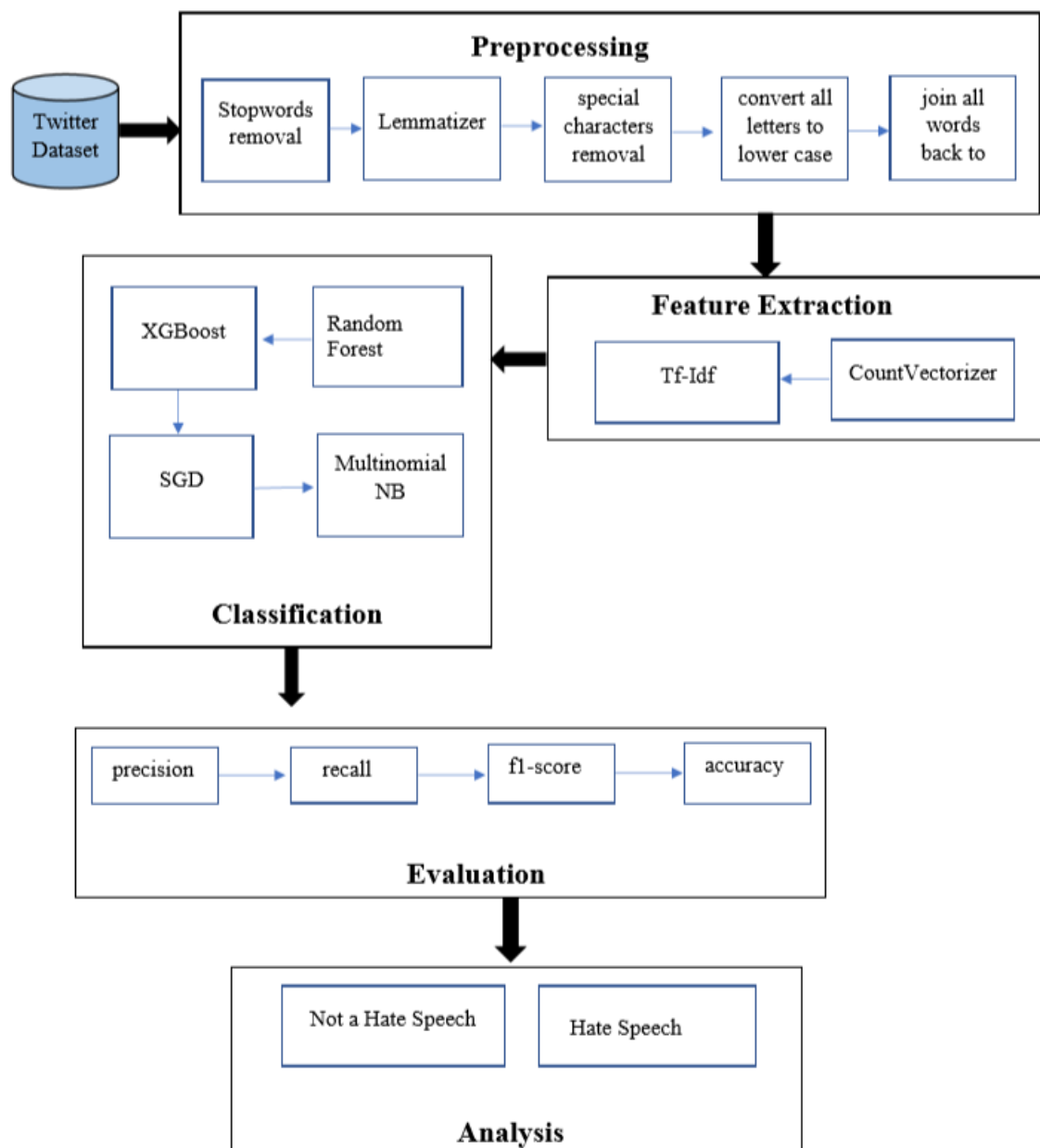
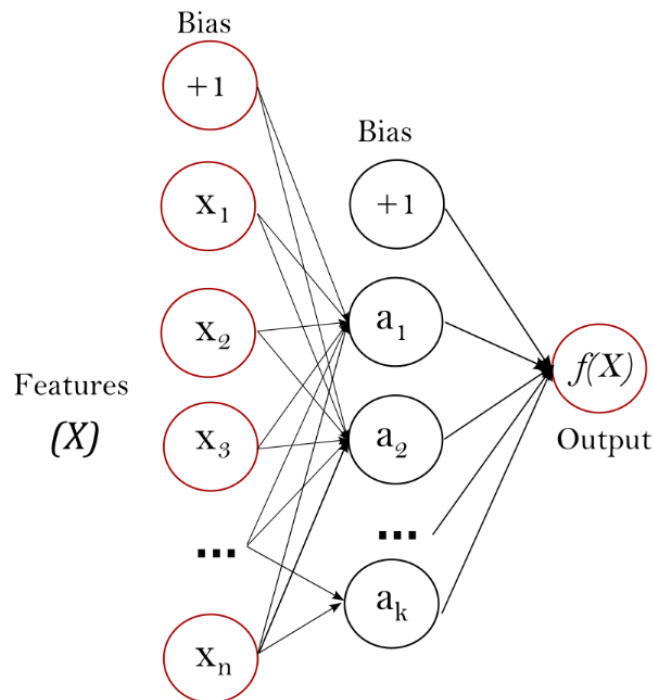


Fig 5.1 Model Architecture

## 5.2 Algorithms Applied

### MLP Classifier:

Input, hidden, and output nodes are the three tiers of nodes in an MLP. With the exception of the input nodes, every node is a neuron with a nonlinear activation function. Backpropagation, a supervised learning technique, is used by MLP for training. Because of its several layers and non-linear activation, MLP isn't a linear perceptron. It can spot data that can't be separated in a linear manner. MLP is a supervised learning technique for learning a function  $f()$ :  $R^m \rightarrow R^o$ , where  $m$  denotes the number of input dimensions and  $o$  signifies the number of output dimensions. Based on a set of features  $X = x_1, x_2, \dots, x_m$  and a target  $y$ , it can learn a non-linear function approximator for classification or regression. One or more non-linear layers, referred to as hidden layers, can be added between the input and output layers, differing from logistic regression. MLP with scalar output and one hidden layer



**Fig 5.1 MLP Classifier**

On the left, the input layer consists of a set of neurons called  $x_i | x_1, x_2, \dots, x_m$  that reflect the input properties. Each neuron in the hidden layer transforms the values from the preceding layer using a weighted linear summation  $w_1x_1 + w_2x_2 + \dots + w_mx_m$ , then applies a non-linear activation function  $g()$ :  $R^R$ , which is comparable to the hyperbolic tan function. The output layer receives the values from the final hidden layer and converts them to output values.

## **Ada-boost:**

Ada-boost, or Adaptive Boosting, is one of the ensemble boosting classifiers. To improve classifier accuracy, it combines several classifiers. Iterative ensemble approach AdaBoost. By combining all of the weak learners into a powerful prediction, the Adaboost technique is a terrific way to improve the performance of a machine learning model. Overfitting may occur if the weak learners are exceedingly weak, and if we dig deeper, we'll find that boosting is extremely difficult to scale. If we want to increase the model's productivity, we must use this method while keeping a few constraints in mind. The AdaBoost classifier combines several low-performing classifiers to produce a high-accuracy strong classifier. The main premise of Adaboost is to train the data sample and establish the weights of classifiers in each iteration so that trustworthy predictions of unusual observations may be made. As a simple classifier, any machine learning algorithm that accepts weights on the training set can be utilized.

## **Xgboost:**

Xgboost is an open-source software program that performs ideally distributed gradient boosting machine learning methods within the Gradient Boosting framework. Xgboost (Extreme Gradient Boosting) is a scalable distributed gradient-boosted decision tree (GBDT) machine learning framework. It contains parallel tree boosting and is the best machine learning tool for regression, classification, and ranking problems. To fully grasp xgboost, you must first comprehend the machine learning concepts and methodologies that it is based on: supervised machine learning, decision trees, ensemble learning, and gradient boosting. Supervised machine learning employs approaches for training a model to find patterns in a dataset with labels and features, and then using the learned model to predict labels on new dataset features. Gradient Boosting Decision Trees (GBDT) is a random forest-like classification and regression decision tree ensemble learning system. Ensemble learning techniques combine various machine learning algorithms to generate a superior model. The gradient boosting approach has been refined to avoid overfitting or bias using parallel processing, tree pruning, missing value management, and regularization.

## Multinomial NB:

The Multinomial Naive Bayes algorithm is a machine learning variation of the Naive Bayes algorithm that excels in multinomial datasets. Because it assesses the likelihood of each label for the input text and then outputs the label with the highest probability, this approach can be used when there are multiple classes to categorize. The following are some of the benefits of using this method for multinomial classification: It is capable of handling large data sets and is straightforward to use with both continuous and discrete data. For training natural language processing models, this is the ideal solution because several labels may be utilized to categorize input.

### 5.3 Training Overview

In this project, the dataset has a large number of hateful speech and fewer numbers of not hateful speech, so overfitting the dataset. Overfitting is a common problem while training a model. This happens when a model performs exceptionally well on the data we used to train it but fails to generalize, previously unseen data points. This can happen for a variety of reasons, including data noise or the model learning to anticipate certain inputs rather than the predictive factors that could aid it in making precise predictions. Overfitting is more likely to occur as a model becomes more complex. Using sklearn's method called `train_test_split()` is used here to split the dataset. Then split the dataset by training part 80% and testing part 20%. Train the tweets about hate tweets or not a hate tweet impact. Then analyze the tweets not hate or hate.

```
# Split data into training and test sets
X = df_oversampled['cleaned_tweets']
y = df_oversampled['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = None)
```

## CODE DEMONSTRATION

### Preprocessing

```
df.drop_duplicates(inplace = True)

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

nltk.download('stopwords')

eng_stops = set(stopwords.words("english"))

lemmatizer = WordNetLemmatizer()

def process_message(review_text):

    # remove all the special characters

    new_review_text = re.sub("[^a-zA-Z]", " ",review_text)

    # convert all letters to lower case

    words = new_review_text.lower().split()

    # remove stop words

    words = [w for w in words if not w in eng_stops]

    # lemmatizer

    words = [lemmatizer.lemmatize(word) for word in words]

    # join all words back to text

    return (" ".join(words))

df['cleaned_tweets']=df['tweet'].apply(lambda x: process_message(x))

df.head()
```

### Feature Extraction

```
count_vect = CountVectorizer(stop_words='english')

transformer = TfidfTransformer(norm='l2', sublinear_tf=True)
```

```
x_train_counts = count_vect.fit_transform(X_train)

x_train_tfidf = transformer.fit_transform(x_train_counts)

x_test_counts = count_vect.transform(X_test)

x_test_tfidf = transformer.transform(x_test_counts)
```

## **Model Building**

### **MLP Model with tf-idf**

```
mlp_tfidf_model = MLPClassifier(max_iter=500, activation='relu',random_state=42)

mlp_tfidf_model.fit(x_train_tfidf,y_train)

mlp_tfidf_predict = mlp_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test,mlp_tfidf_predict))

print (classification_report(y_test, mlp_tfidf_predict))
```

### **MLP Model with CV**

```
mlp_cv_model =MLPClassifier(max_iter=500, activation='relu',random_state=42)

mlp_cv_model.fit(x_train_cv,y_train)

mlp_cv_predict = mlp_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test,mlp_cv_predict))

print (classification_report(y_test, mlp_cv_predict))
```

### **AdaBoost Model with tf-idf**

```
ad_tfidf_model = AdaBoostClassifier(n_estimators=500, random_state=42)

ad_tfidf_model.fit(x_train_tfidf,y_train)

ad_tfidf_predict = ad_tfidf_model.predict(x_test_tfidf)

print (confusion_matrix(y_test,ad_tfidf_predict))

print (classification_report(y_test, ad_tfidf_predict))
```



### **AdaBoost Model with CV**

```
ad_cv_model = AdaBoostClassifier(n_estimators=500, random_state=42)

ad_cv_model.fit(x_train_cv,y_train)

ad_cv_predict = ad_cv_model.predict(x_test_cv)

print (confusion_matrix(y_test,ad_cv_predict))

print (classification_report(y_test, ad_cv_predict))
```

### **XGBoost Model with tf-idf**

```
xgb_tfidf_model = XGBClassifier(random_state=42, learning_rate=0.9)

xgb_tfidf_model.fit(x_train_tfidf,y_train)

xgb_tfidf_predict = xgb_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test,xgb_tfidf_predict))

print(classification_report(y_test,xgb_tfidf_predict))
```

### **XGBoost Model with CV**

```
xgb_cv_model = XGBClassifier(random_state=42, learning_rate=0.9)

xgb_cv_model.fit(x_train_cv,y_train)

xgb_cv_predict = xgb_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test,xgb_cv_predict))

print(classification_report(y_test,xgb_cv_predict))
```

### **MultinomialNB Model with CV**

```
nb_cv_model = MultinomialNB()

nb_cv_model.fit(x_train_cv, y_train)

nb_cv_predict = nb_cv_model.predict(x_test_cv)
```

```
print(confusion_matrix(y_test,nb_cv_predict))  
  
print (classification_report(y_test, nb_cv_predict))
```

### **MultinomialNB Model with Tfidf**

```
nb_tfidf_model = MultinomialNB()  
  
nb_tfidf_model.fit(x_train_tfidf, y_train)  
  
nb_tfidf_predict = nb_tfidf_model.predict(x_test_tfidf)  
  
print(confusion_matrix(y_test,nb_tfidf_predict))  
  
print (classification_report(y_test, nb_tfidf_predict))
```

## Chapter 6

### EXPERIMENTAL DESIGN AND EVALUATION

Accuracy, Precision, Recall, and F1 scores are used to evaluate the system's performance.

#### Accuracy:

When evaluating classification models, accuracy is one parameter to consider. Our model's accuracy is expressed as a percentage of correct predictions.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

The following formula can be used to calculate accuracy in terms of positives and negatives for binary classification:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$TP$  is represented as True Positives,  $TN$  is represented as True Negatives,  $FP$  is represented as False Positives and  $FN$  is represented as False Negatives.

#### Precision:

Precision relates to how closely identical measurements of the same thing are. Precision is distinct from accuracy.

$$\text{Precision} = \frac{TP}{TP + FP}$$

#### Recall:

The recall is a classification model's capacity to recognize all data points in a relevant class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

#### F1 score:

The harmonic mean of precision and recall is called the F1 score. It is a single number that combines precision and recall.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

## ROC Curve:

A receiver operating characteristic curve (ROC curve) is a graph that depicts a classification model's performance across all categorization levels. The ROC curve is a performance metric for classification tasks at different threshold levels. The receiver operating characteristic (ROC) curve is commonly used to assess the performance of binary classification techniques. Instead of a single value like most other metrics, it displays a graphical representation of a classifier's performance. Two parameters are plotted in this curve:

- ☐ True Positive Rate
- ☐ False Positive Rate

The TPR is the chance that a true positive will test positive.

$$TPR = \frac{TP}{TP + FN}$$

The fraction of true negatives misclassified as positives are known as the false-positive rate.

$$FPR = \frac{FP}{FP + TN}$$

## 6.1 Experimental Design

In this project, I have done three experiments for analyzing purposes. Here the first experiment is the NLP technique count vectorizer with four algorithms they are MLP algorithm is a Deep Learning Algorithm then three ML algorithms are Adaboost, Xgboost and Multinomial NB algorithms are used. Then the second experiment is the NLP technique Tfidf transformer with four algorithms explained above. The third experiment is ROC curve classifiers of NLP techniques Tfidf transformer and CV.

## Experiment-1: Count Vectorizer with models MLP, Adaboost, Xgboost and Multinomial NB

### CV with Accuracy Score

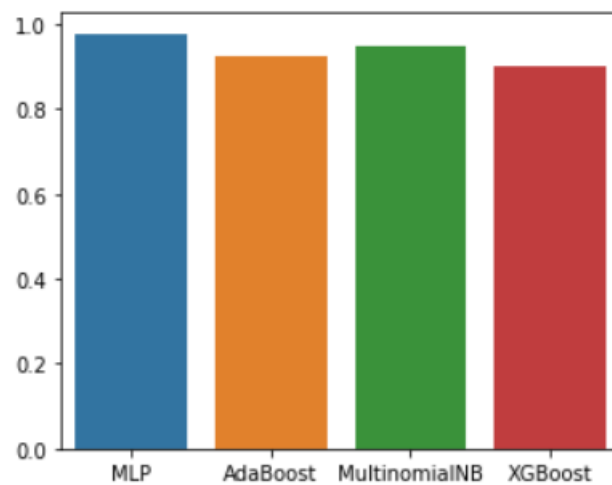


Fig 6.1.1 CV with Accuracy Score

### CV with F1 Score

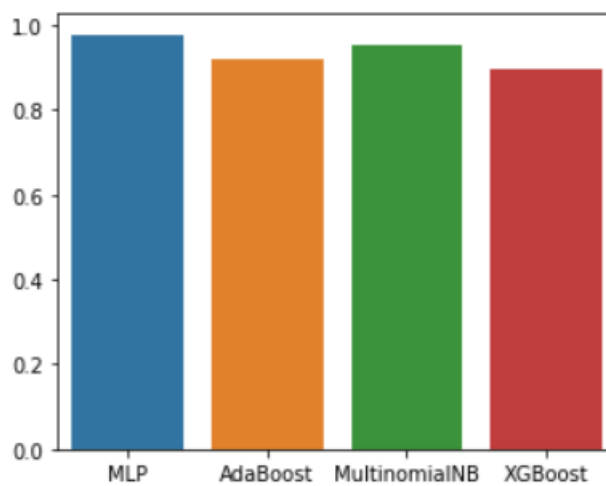
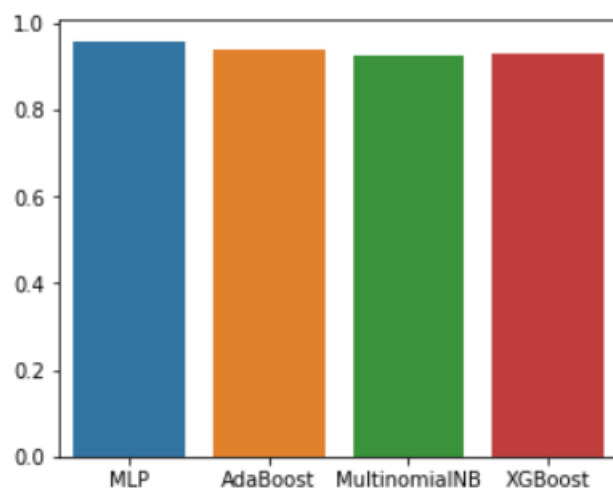


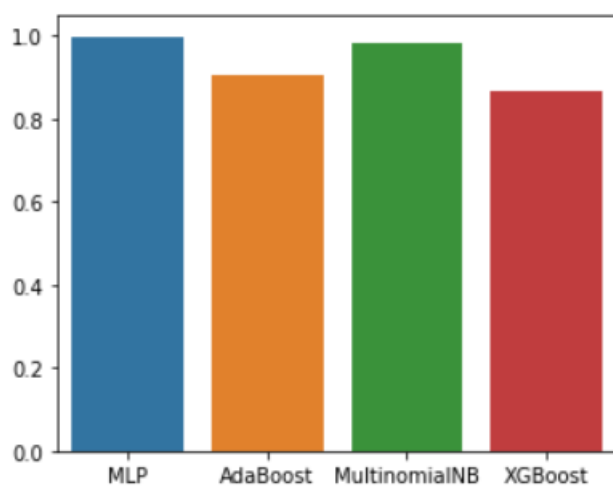
Fig 6.1.2 CV with F1 Score

### CV with Precision Score



**Fig 6.1.3 CV with Precision Score**

### CV with Recall Score



**Fig 6.1.4 CV with Recall Score**

## Experiment-2: Tf-Idf Transformer with models MLP, Adaboost, Xgboost and Multinomial NB

### Tf-Idf with Accuracy Score

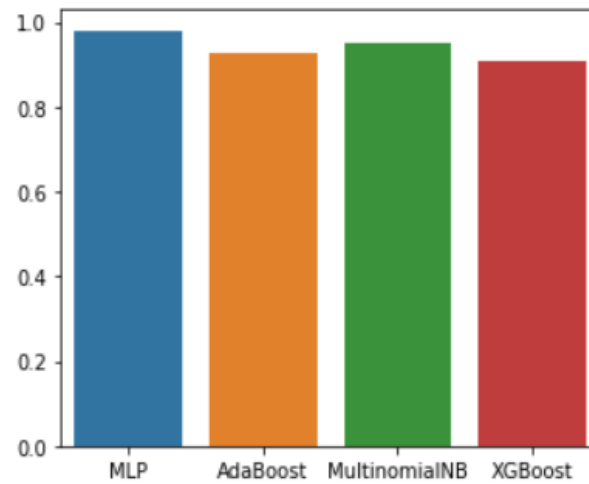


Fig 6.1.5 Tf-Idf with Accuracy Score

### Tf-Idf with F1 Score

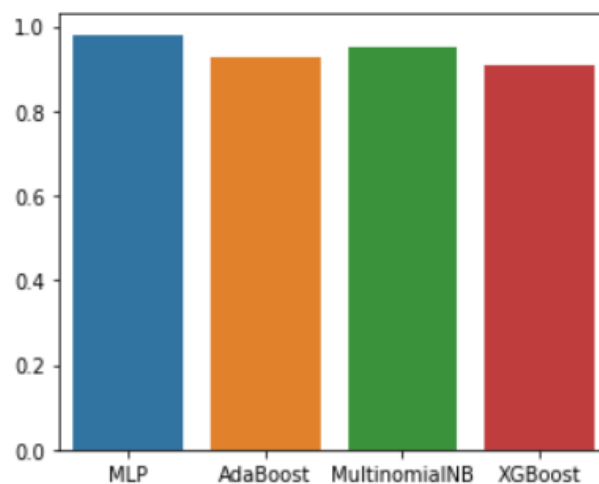
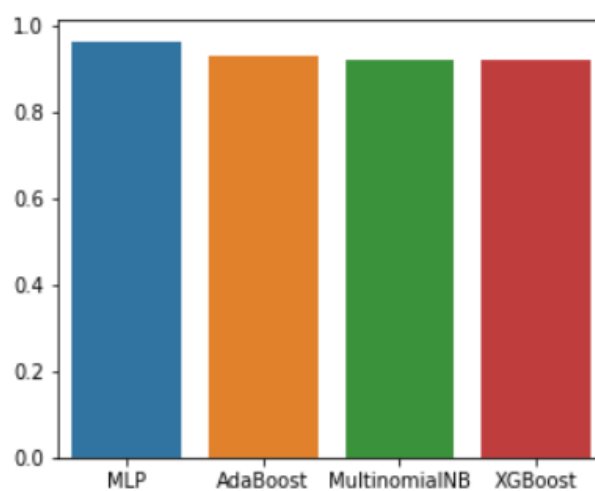


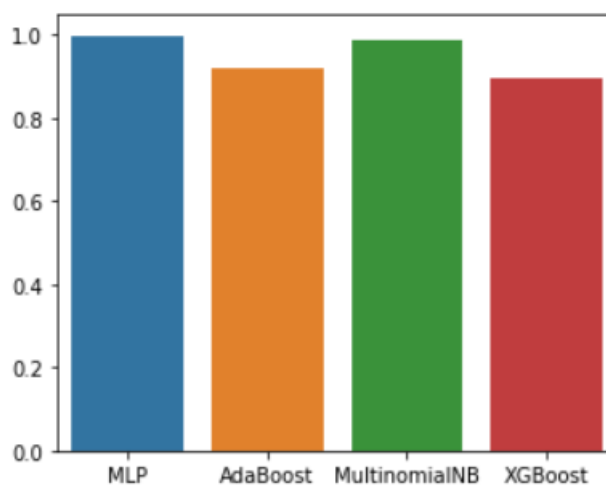
Fig 6.1.6 Tf-Idf with F1 Score

### Tf-Idf with Precision Score



**Fig 6.1.7 Tf-Idf with Precision Score**

### Tf-Idf with Recall Score



**Fig 6.1.8 Tf-Idf with Recall Score**



### Experiment-3: ROC Curve Classifiers of CV and Tf-Idf

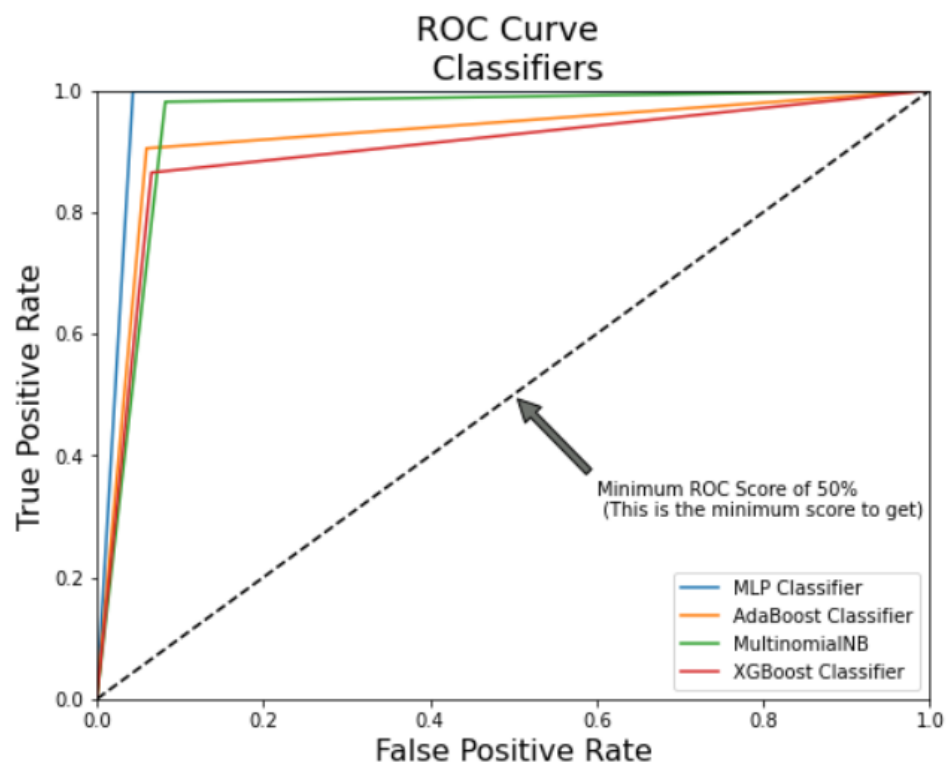


Fig 6.1.9 ROC Curve Classifiers of CV

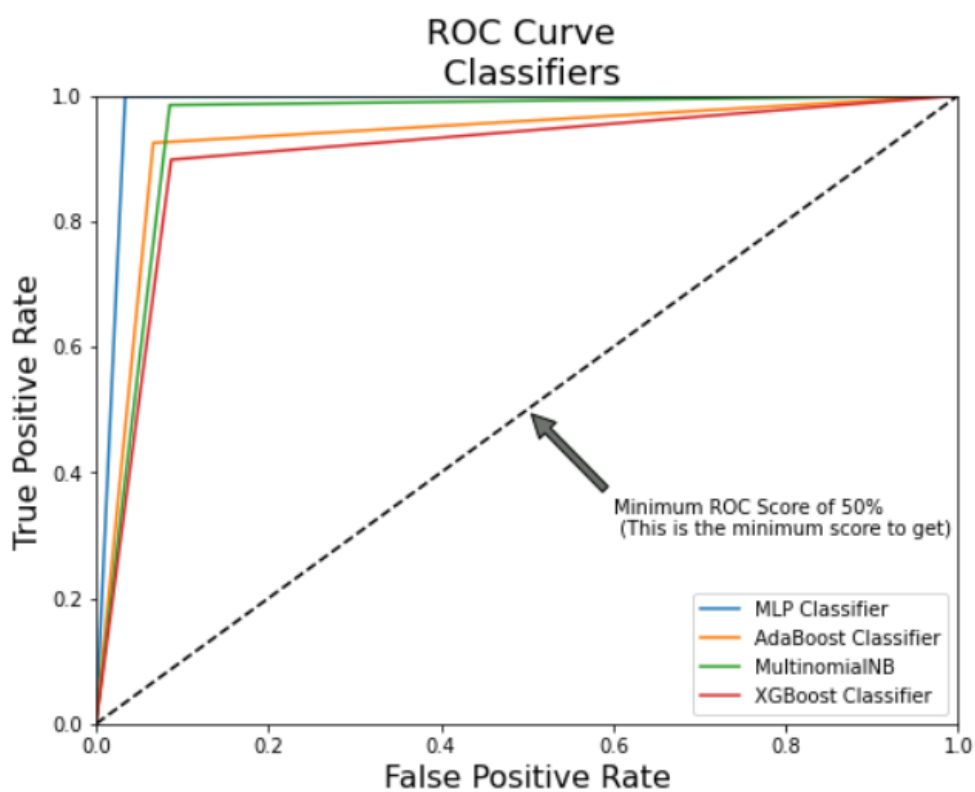


Fig 6.1.10 ROC Curve Classifiers of Tf-Idf

## 6.2 Experimental Results

The experimental results will be evaluated by accuracy score which is the metric used for the binary classification. Accuracy is defined as the percentage of correct classifications among all categories. The most straightforward performance statistic is accuracy, which is just the proportion of correctly predicted observations to all observations.

### Experiment-1: Count Vectorizer with models MLP, Adaboost, Xgboost and Multinomial NB

Table 6.2.1 Experimental-1

Algorithms with CV	Accuracy
MLP	<b>0.98</b>
Adaboost	0.92
Xgboost	0.90
Multinomial NB	0.95

### Experiment-2: Tf-Idf Transformer with models MLP, Adaboost, Xgboost and Multinomial NB

Table 6.2.2 Experimental-2

Algorithms with TFIDF	Accuracy
MLP	<b>0.98</b>
Adaboost	0.93
Xgboost	0.90
Multinomial NB	0.95

### Experiment-3: ROC Curve Classifiers of CV and Tf-Idf

Table 6.2.3 Experimental-3

Algorithms with TFIDF	Accuracy	
	CV	Tf-Idf
MLP	<b>0.97</b>	<b>0.98</b>
Adaboost	0.92	0.92
Xgboost	0.89	0.90
Multinomial NB	0.94	0.94

### 6.3 Customer Evaluation and Feedback

When the product is given to the customer, a one-time developer performs a demo, and the customer reviews the product to see if it meets their expectations. During the evaluation, the customer evaluates needs, such as designing styles on pages and then checks the demo. If the product does not fulfill the customer's expectations, the customer provides feedback, and the developer responds quickly. And, assuming you meet the customer's needs and receive positive feedback, what improvements do you think you should make in the future product.

## Chapter 7

### MODEL OPTIMIZATION

#### 7.1 Overview of Model Tuning and Best Parameters Selection

A feedforward artificial neural network model, the multilayer perceptron (MLP), maps input data sets to a collection of acceptable outputs. An MLP is made up of numerous layers, each of which is fully connected to the one before it. Except for the nodes of the input layer, the nodes of the layers are neurons with nonlinear activation functions. One or more nonlinear hidden layers may exist between the input and output layers.

- **Hidden layer sizes:** We can specify the number of layers and nodes in the Neural Network classifier using this option. The number of nodes at the  $i$ th position is represented by each element in the tuple, with  $I$  being the tuple's index. As a result, the tuple's length represents the total number of hidden layers in the neural network.
- **Max iter:** The number of epochs is shown.
- **Activation:** The hidden layers activation function.
- **Solver:** This parameter defines the weight optimization technique for the nodes.

#### 7.2 Model Tuning Process and Experiments

**default=None, test size float or int**

If you use float, the value should be between 0.0 and 1.0 and represent the percentage of the dataset that should be included in the train split. The absolute number of test samples is represented by int. The complement of the train size is used if the value is None. It will be 0.25 if train size is also None.

**default=None, train\_size float or int**

If float, it should be in the range of 0.0 to 1.0 and signify the proportion of the dataset that should be included in the train split. If int is used, it represents the total number of train samples. If None is specified, the value is set to the complement of the test size.

**default=None, Stratify: array-like**

If None is selected, these are used as class labels to stratify the data.

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = None)
```

**list, default=None, stop\_words: {'english'}**

If 'English' is specified, an English stop word list is utilized. There are various recognized problems with 'English,' therefore you should think about switching.

**default='l2', Norm: {'l1', 'l2'}**

The unit norm for each output row will be either: 'l2': The sum of vector elements' squares is one. When the l2 norm is used, the cosine similarity between two vectors equals their dot product. 'l1': The sum of vector elements' absolute values is 1.

**default=False, sublinear\_tf :bool**

Sublinear tf scaling is done by replacing tf with  $1 + \log(\text{tf})$ .

```
count_vect = CountVectorizer(stop_words='english')
transformer = TfidfTransformer(norm='l2', sublinear_tf=True)
```

**default=200, max\_iter: int**

The number of epochs is shown.

**default='relu', activation: {'relu'}**

The hidden layers activation function.

- 'relu' -is a rectified linear unit function

**default=None, random\_state: int, RandomState instance**

When solver='sgd' or 'adam', it decides random number generation for weights and bias initialization, train-test split if early stopping is used, and batch sampling.

```
mlp_cv_model = MLPClassifier(max_iter=500, activation='relu', random_state=42)
```

**default=50, n\_estimators: int**

The number of estimators at which boosting is turned off. The learning process is halted early if the fit is ideal.

**RandomState instance or None, default=None, random\_state: int**

Each boosting iteration's random seed is controlled by this variable. As a result, it's only used when the base estimator reveals a random state.

```
ad_cv_model = AdaBoostClassifier(n_estimators=500, random_state=42)
```

**default=0.1, learning\_rate: float,**

The contribution of each tree is reduced by the learning rate. Between learning rate and n estimators, there is a trade-off.

**random\_state: int, default=None, RandomState instance or None**

Controls the random seed provided to each Tree estimator during each boosting cycle.

```
xgb_cv_model = XGBClassifier(random_state=42, learning_rate=0.9)
```

**Thresholds: ndarray of shape = (n\_thresholds,)**

Reduce the decision function's thresholds for computing fpr and tpr.

```
mlp_fpr, mlp_tpr, threshold = roc_curve(y_test, mlp_tfidf_predict)
ad_fpr, ad_tpr, threshold = roc_curve(y_test, ad_tfidf_predict)
nb_fpr, nb_tpr, threshold = roc_curve(y_test, nb_tfidf_predict)
xgb_fpr, xgb_tpr, threshold = roc_curve(y_test, xgb_tfidf_predict)
```

## Chapter 8

### PRODUCT DELIVERY AND DEPLOYMENT

#### 8.1 User Manuals

##### **Need for the User manual:**

Developers clearly supply the product after developing the proposed system. The user handbook is required upon delivery. Because they are end-users, the user has no previous experience with the product. If a problem arises while using this product, the user can quickly resolve the problem by consulting the user manual. The first step we have to enter the tweet. Then predict the given tweet for the prediction I used the best accuracy model **MLP classifier**. Next result analysis whether the given tweet is a hate tweet or not a hate tweet.

#### 8.2 Delivery Schedule

Sentiment analysis using Twitter datasets using machine learning and deep learning can be deployed and delivered inside the product delivery schedule.

**Table 8.2 Delivery Schedule**

<b>Date and No of Days</b>	<b>Which one is Deliver the customer (user)</b>
First week (3-4 days)	Project Proposal
Second week	Designing
Third week	Users accept the design and then develop code
Fifth week	Styling changes in Design
Sixth week	Run in local server
Seventh week	Complete design and Processing
Tenth week	solving user Problem
Final Stage	Product Delivery in Market

### **8.3 Deployment Process**

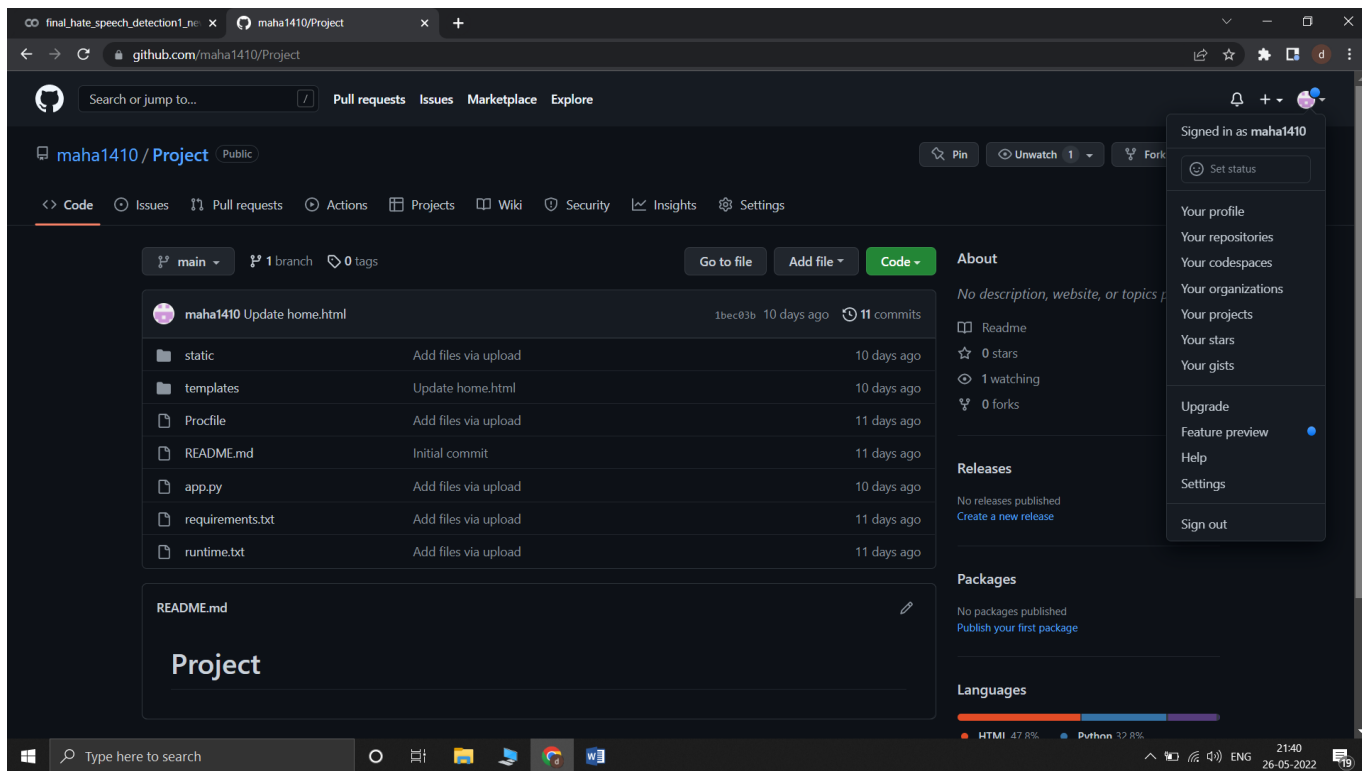
In this project, I deploy the best accuracy model MLP classifier using the Flask Framework. The site's back end is powered by Python code. The aim to deploy the process is to analyze the text. By flask framework, I have created a flask web app to predict whether the given text is hate tweets or not a hate tweets.

#### **Flask Framework:**

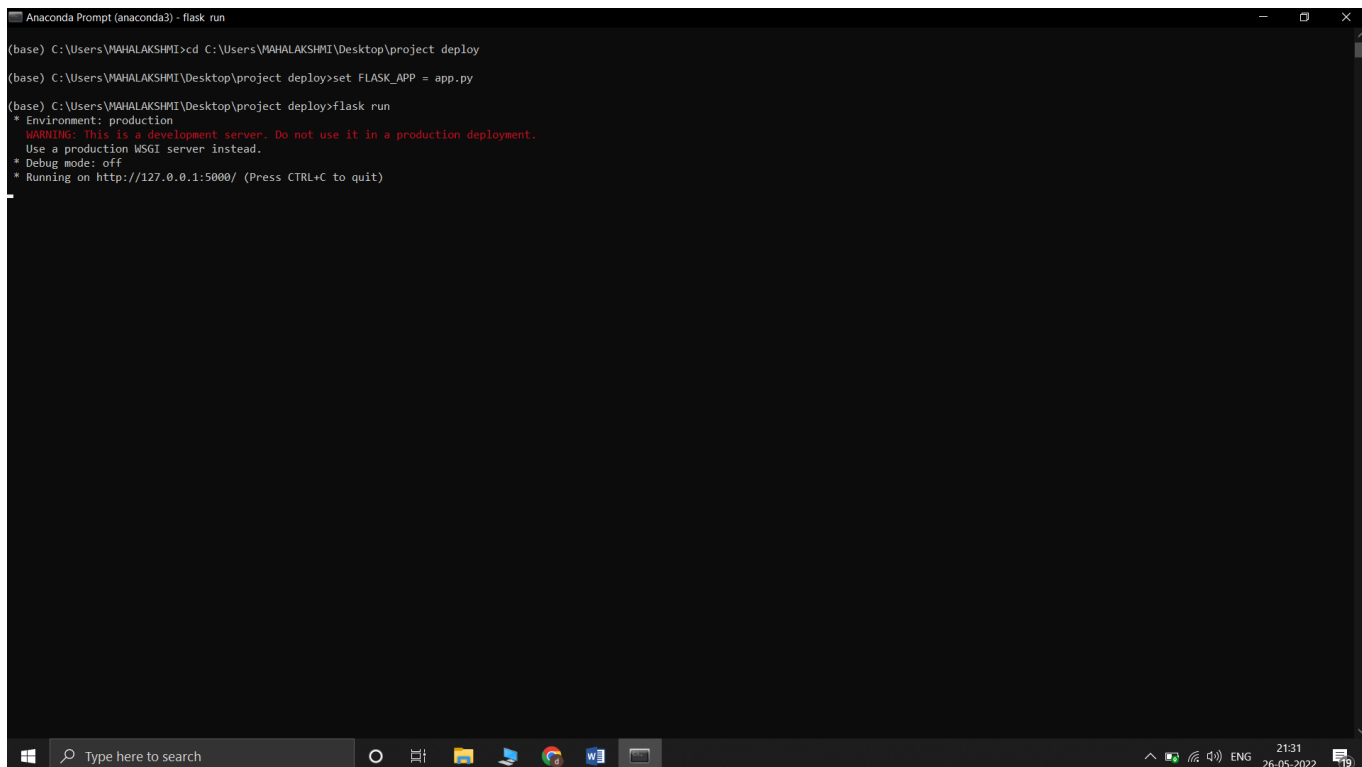
Flask is a simple and lightweight Python web framework that provides essential tools and capabilities for developing online applications in Python. Because you can build a web application rapidly using only a single Python file, it allows developers more flexibility and is more accessible to new developers. Flask is also extensible, requiring no specific directory structure or boilerplate code to get started.

Github Link: <https://github.com/maha1410/Project>





URL Link: <http://127.0.0.1:5000/>



## **Chapter 9**

# **CONCLUSION**

### **9.1 Summary**

The summary of the project is the proposed model for hate speech detection that analyze comments to identify hate or not a hate speech using ML and DL algorithms. Then collect the data set, next step is preprocessing the dataset then only we get the best accuracy score. The next step is feature extraction, we used two NLP techniques Count Vectorizer and MLP, Adaboost, Xgboost, and Multinomial NB with Tf-Idf transformers that use ML and DL techniques. Then compare the models by the accuracy score the best model is the MLP classifier with Tf-Idf. It outperforms all other algorithms. In both the NLP technique (Tf-Idf) with MLP classifier, it shows the best result in accuracy score is 0.99. The dataset is larger which is a benefit for the DL algorithm for the best performance than other ML algorithms.

### **9.2 Limitations and Future Work**

This proposed system has some limitations such as providing the results for trained datasets only. In addition, sentiment analytics for textual datasets are available because of system configuration, it takes longer to execute during training. In the future we extend the framework to classify the tweets with various deep learning algorithms and detecting hate speech from big data is a very challenge. Then also analyze the various types of tweets such as images, video, audio or emoticons.

## REFERENCES

- [1] Z. Al-Makhadmeh and A. Tolba, “Automatic hate speech detection using killer natural language processing optimizing ensemble deep learning approach,” *Computing*, vol. 102, no. 2, pp. 501–522, Feb. 2020.
- [2] Gaydhani A, Doma V, Kendre S, Bhagwat L (2018) Detecting hate speech and offensive language on Twitter using machine learning: an N-gram and TFIDF based approach. *arXiv preprint arXiv:1809.08651*.
- [3] Kristiawan Nugroho, Edy Noersasongko, Purwanto, Muljono, Ahmad Zainul Fanani, Affandy, Ruri Suko Basuki, “Improving Random Forest Method to Detect Hate speech and Offensive Word” ©2019 IEEE, International Conference on Information and Communications Technology (ICOIACT).
- [4] Lin Jiang, Yoshimi Suzuki, “Detecting hate speech from tweets for sentiment analysis”, ©2019 IEEE, 6th International Conference on Systems and Informatics (ICSAI).
- [5] Bhavesh Pariyani, Krish Shah, Meet Shah, Tarjini Vyas, Sheshang Degadwala, “Hate Speech Detection in Twitter using Natural Language Processing”, *Proceedings of the Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV 2021)*. IEEE Xplore Part Number: CFP21ONG-ART; 978-0-7381-1183-4.
- [6] Ricardo Martins, Marco Gomes, Jos’e Jo ~ ao Almeida, Paulo Novais, Pedro Henriques, “Hate speech classification in social media using emotional analysis”, 7th Brazilian Conference on Intelligent Systems ©2018 IEEE DOI 10.1109/BRACIS.2018.00019.
- [7] Hitesh Kumar Sharma, K Kshiti, Shailendra, “NLP and Machine Learning Techniques for Detecting Insulting Comments on Social Networking Platforms”, *International Conference on Advances in Computing and Communication Engineering (ICACCE-2018)* Paris, France 22-23 June 2018.
- [8] Sujatha Arun Kokatnoor, Balachandran Krishnan, “Twitter Hate Speech Detection using Stacked Weighted Ensemble (SWE) Model”, ©2020 IEEE Fifth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN).
- [9] Junanda Patihullah, Edi Winarko, “Hate Speech Detection for Indonesia Tweets Using Word Embedding and Gated Recurrent Unit”, *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)* Vol.13, No.1, January 2019, pp. 43~52, DOI: <https://doi.org/10.22146/ijccs.40125>.

- [10] R. Cao, R. K.-W. Lee, and T.-A. Hoang, “DeepHate: Hate speech detection via multi-faceted text representations,” in Proc. 12th ACM Conf. Web Sci., Southampton, U.K., Jul. 2020, pp. 11–20.
- [11] T. Davidson, D. Warmesley, M. Macy, and I. Weber, “Automated hate speech detection and the problem of offensive language,” in Proc. ICWSM, Montreal, QC, Canada, May 2017, pp. 15–18.
- [12] ] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma, “Deep learning for hate speech detection in tweets,” in Proc. 26th Int. Conf. World Wide Web Companion (WWW Companion), Perth, WA, Australia, Apr. 2017, pp. 759–760.
- [13] P. Burnap and M. L. Williams, “Cyber hate speech on Twitter: An application of machine classification and statistical modeling for policy and decision making,” Policy Internet, vol. 7, no. 2, pp. 223–242, Jun. 2015.
- [14] M. O. Ibrohim and I. Budi, “Multi-label hate speech and abusive language detection in Indonesian Twitter,” in Proc. 3rd Workshop Abusive Lang. Online, Florence, Italy, Aug. 2019, pp. 46–57.
- [15] V. Basile, C. Bosco, E. Fersini, “SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in Twitter,” in Proc. 13th Int. Workshop Semantic Eval., Minneapolis, MN, USA, Jun. 2019, pp. 54–63.
- [16] I. Alfina, R. Mulia, M. I. Fanany, and Y. Ekanata, “Hate speech detection in the Indonesian language: A dataset and preliminary study,” Oct. 2017, pp. 233–238
- [17] M. Mozafari, R. Farahbakhsh, and N. Crespi, “Hate speech detection and racial bias mitigation in social media based on BERT model,” PLoS ONE, vol. 15, no. 8, pp. 1–26, Aug. 2020.
- [18] M. Wiegand, J. Ruppenhofer, and T. Kleinbauer, “Detection of abusive language: The problem of biased datasets,” in Proc. HLT-NAACL, Minneapolis, MN, USA, Jun. 2019, pp. 602–608.

## APPENDIX-A

## DATA SET

The id, label, and tweet datasets for this proposed system were taken from the GitHub dataset. This dataset has the text id, whereas tweet is a text dataset that contains the tweets. Then give the primary content with its own set of values. Finally, assign a class label to hate tweets, such as '1' and not hate tweets, such as '0'. In the dataset, they are responsible for a total of 29720 non-hate tweets and 2242 hate tweets.

[illegible]

## APPENDIX-B

### SOURCE CODE

#### **## Imported libraries**

```
import warnings

warnings.filterwarnings('ignore')

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from nltk.stem import WordNetLemmatizer

from sklearn.utils import resample

from PIL import Image

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, f1_score

from sklearn.naive_bayes import MultinomialNB

from sklearn.ensemble import AdaBoostClassifier

from xgboost import XGBClassifier

from sklearn.linear_model import SGDClassifier

from sklearn import metrics, utils

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.metrics import roc_curve
```

```

from sklearn.metrics import roc_auc_score

import re

import nltk

nltk.download('stopwords')

from wordcloud import WordCloud, STOPWORDS

from nltk.corpus import stopwords

import nltk

nltk.download('wordnet')

df = pd.read_csv("train_tweets.csv")

df.head()

hate_tweet = df[df.label == 1]

hate_tweet.head()

nothate_tweet = df[df.label == 0]

nothate_tweet.head()

```

## **# Data Preprocessing**

```

df.drop_duplicates(inplace = True)

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

nltk.download('stopwords')

eng_stops = set(stopwords.words("english"))

lemmatizer = WordNetLemmatizer()

def process_message(review_text):

    # remove all the special characters

    new_review_text = re.sub("[^a-zA-Z]", " ",review_text)

    # convert all letters to lower case

    words = new_review_text.lower().split()

```

```

# remove stop words

words = [w for w in words if not w in eng_stops]

# lemmatizer

words = [lemmatizer.lemmatize(word) for word in words]

# join all words back to text

return (" ".join(words))

df['cleaned_tweets']=df['tweet'].apply(lambda x: process_message(x))

df.head()

```

## **## Visualization**

### **# Hate Word clouds**

```

text = " ".join(review for review in hate_tweet.tweet)

wordcloud = WordCloud(max_font_size=50, max_words=100,
background_color="white").generate(text)

fig = plt.figure(figsize = (20, 6))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis("off")

plt.show()

```

### **# Not a hate Word clouds**

```

from PIL import Image

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

text = " ".join(review for review in nothate_tweet.tweet)

wordcloud = WordCloud(max_font_size=50, max_words=100,
background_color="white").generate(text)

fig = plt.figure(figsize = (20, 6))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis("off")

```



```
plt.show()
```

## **# Handling Imbalanced data**

```
fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])
```

```
langs = ['Not a HateSpeech','HateSpeech']
```

```
data = [len(df[df.label==0]),len(df[df.label==1])]
```

```
ax.bar(langs,data)
```

```
plt.show()
```

## **## Over Sampling**

```
df_majority = df[df.label==0]
```

```
df_minority = df[df.label==1]
```

```
df_minority_oversampled = resample (df_minority,
```

```
                                replace=True,
```

```
                                n_samples=len(df_majority),
```

```
                                random_state=123)
```

```
df_oversampled = pd.concat([df_minority_oversampled, df_majority])
```

```
df_oversampled['label'].value_counts()
```

```
fig = plt.figure()
```

```
ax = fig.add_axes([0,0,1,1])
```

```
langs = ['Not a HateSpeech','HateSpeech']
```

```
data = [len(df_oversampled[df_oversampled.label==0]),
```

```
len(df_oversampled[df_oversampled.label==1])]
```

```
ax.bar(langs,data)
```

```
plt.show()
```

## **## Splitting the data**

```
# Split data into training and test sets

X = df_oversampled['cleaned_tweets']

y = df_oversampled['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = None)
```

## **# Feature Extraction**

### **## CountVectorizer**

### **## Tfidf**

```
count_vect = CountVectorizer(stop_words='english')

transformer = TfidfTransformer(norm='l2',sublinear_tf=True)

x_train_cv = count_vect.fit_transform(X_train)

x_train_tfidf = transformer.fit_transform(x_train_cv)

print(x_train_cv.shape)

print(x_train_tfidf.shape)

x_test_cv = count_vect.transform(X_test)

x_test_tfidf = transformer.transform(x_test_cv)

print(x_test_cv.shape)

print(x_test_tfidf.shape)
```

## **# Model Building**

### **## MLP Model with CV**

```
mlp_cv_model = MLPClassifier(max_iter=500, activation='relu',random_state=42)

mlp_cv_model.fit(x_train_cv,y_train)

mlp_cv_predict = mlp_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test,mlp_cv_predict))

print(classification_report(y_test, mlp_cv_predict))

f1_mlp_cv = f1_score(y_test, mlp_cv_predict)
```

```
pr_mlp_cv = metrics.precision_score(y_test, mlp_cv_predict)

rc_mlp_cv = metrics.recall_score(y_test, mlp_cv_predict)

acc_mlp_cv = metrics.accuracy_score(y_test, mlp_cv_predict)
```

### **## AdaBoost Model with CV**

```
ad_cv_model = AdaBoostClassifier(n_estimators=500, random_state=42)

ad_cv_model.fit(x_train_cv, y_train)

ad_cv_predict = ad_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test, ad_cv_predict))

print(classification_report(y_test, ad_cv_predict))

f1_ad_cv = f1_score(y_test, ad_cv_predict)

pr_ad_cv = metrics.precision_score(y_test, ad_cv_predict)

rc_ad_cv = metrics.recall_score(y_test, ad_cv_predict)

acc_ad_cv = metrics.accuracy_score(y_test, ad_cv_predict)
```

### **## XGBoost Model with CV**

```
xgb_cv_model = XGBClassifier(random_state=42, learning_rate=0.9)

xgb_cv_model.fit(x_train_cv, y_train)

xgb_cv_predict = xgb_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test, xgb_cv_predict))

print(classification_report(y_test, xgb_cv_predict))

f1_xgb_cv = f1_score(y_test, xgb_cv_predict)

pr_xgb_cv = metrics.precision_score(y_test, xgb_cv_predict)

rc_xgb_cv = metrics.recall_score(y_test, xgb_cv_predict)

acc_xgb_cv = metrics.accuracy_score(y_test, xgb_cv_predict)
```

### **## MultinomialNB Model with CV**

```
nb_cv_model = MultinomialNB()
```

```

nb_cv_model.fit(x_train_cv, y_train)

nb_cv_predict = nb_cv_model.predict(x_test_cv)

print(confusion_matrix(y_test,nb_cv_predict))

print(classification_report(y_test, nb_cv_predict))

f1_nb_cv = f1_score(y_test, nb_cv_predict)

pr_nb_cv = metrics.precision_score(y_test, nb_cv_predict)

rc_nb_cv = metrics.recall_score(y_test, nb_cv_predict)

acc_nb_cv = metrics.accuracy_score(y_test, nb_cv_predict)

```

### **##Comparison of cv models with F1 scores,accuracy,precision and recall**

```

cv_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

f1_cv_models = [f1_mlp_cv,f1_ad_cv, f1_nb_cv, f1_xgb_cv]

plt.figure(figsize=(14,4))

sns.barplot(x = cv_models, y= f1_cv_models)

cv_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

pr_cv_models = [pr_mlp_cv,pr_ad_cv, pr_nb_cv, pr_xgb_cv]

plt.figure(figsize=(14,4))

sns.barplot(x = cv_models, y= pr_cv_models)

cv_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

rc_cv_models = [rc_mlp_cv,rc_ad_cv, rc_nb_cv, rc_xgb_cv]

plt.figure(figsize=(14,4))

sns.barplot(x = cv_models, y= rc_cv_models)

cv_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

acc_cv_models = [acc_mlp_cv,acc_ad_cv, acc_nb_cv, acc_xgb_cv]

plt.figure(figsize=(14,4))

sns.barplot(x = cv_models, y= acc_cv_models)

```

## **## MLP Model with tfidf**

```
mlp_tfidf_model = MLPClassifier(max_iter=500, activation='relu', random_state=42)

mlp_tfidf_model.fit(x_train_tfidf, y_train)

mlp_tfidf_predict = mlp_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test, mlp_tfidf_predict))

print(classification_report(y_test, mlp_tfidf_predict))

f1_mlp_tfidf = f1_score(y_test, mlp_tfidf_predict)

pr_mlp_tfidf = metrics.precision_score(y_test, mlp_tfidf_predict)

rc_mlp_tfidf = metrics.recall_score(y_test, mlp_tfidf_predict)

acc_mlp_tfidf = metrics.accuracy_score(y_test, mlp_tfidf_predict)
```

## **## AdaBoost Model with tf-idf**

```
ad_tfidf_model = AdaBoostClassifier(n_estimators=500, random_state=42)

ad_tfidf_model.fit(x_train_tfidf, y_train)

ad_tfidf_predict = ad_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test, ad_tfidf_predict))

print(classification_report(y_test, ad_tfidf_predict))

f1_ad_tfidf = f1_score(y_test, ad_tfidf_predict)

pr_ad_tfidf = metrics.precision_score(y_test, ad_tfidf_predict)

rc_ad_tfidf = metrics.recall_score(y_test, ad_tfidf_predict)

acc_ad_tfidf = metrics.accuracy_score(y_test, ad_tfidf_predict)
```

## **## XGBoost Model with tf-idf**

```
xgb_tfidf_model = XGBClassifier(random_state=42, learning_rate=0.9)

xgb_tfidf_model.fit(x_train_tfidf, y_train)

xgb_tfidf_predict = xgb_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test, xgb_tfidf_predict))
```

```

print(classification_report(y_test,xgb_tfidf_predict))

f1_xgb_tfidf = f1_score(y_test, xgb_tfidf_predict)

pr_xgb_tfidf = metrics.precision_score(y_test, xgb_tfidf_predict)

rc_xgb_tfidf = metrics.recall_score(y_test, xgb_tfidf_predict)

acc_xgb_tfidf = metrics.accuracy_score(y_test, xgb_tfidf_predict)

```

## **## MultinomialNB Model with tf-idf**

```

nb_tfidf_model = MultinomialNB()

nb_tfidf_model.fit(x_train_tfidf, y_train)

nb_tfidf_predict = nb_tfidf_model.predict(x_test_tfidf)

print(confusion_matrix(y_test,nb_tfidf_predict))

print(classification_report(y_test, nb_tfidf_predict))

f1_nb_tfidf = f1_score(y_test, nb_tfidf_predict)

pr_nb_tfidf = metrics.precision_score(y_test, nb_tfidf_predict)

rc_nb_tfidf = metrics.recall_score(y_test, nb_tfidf_predict)

acc_nb_tfidf = metrics.accuracy_score(y_test, nb_tfidf_predict)

```

## **## Comparison of tfidf models with F1 scores,accuracy,precision and recall**

```

tfidf_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

f1_tfidf_models = [f1_mlp_tfidf,f1_ad_tfidf, f1_nb_tfidf, f1_xgb_tfidf]

plt.figure(figsize=(14,4))

sns.barplot(x = tfidf_models, y= f1_tfidf_models)

tfidf_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

pr_tfidf_models = [pr_mlp_tfidf,pr_ad_tfidf, pr_nb_tfidf, pr_xgb_tfidf]

plt.figure(figsize=(14,4))

sns.barplot(x = tfidf_models, y= pr_tfidf_models)

tfidf_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

```

```

rc_tfidf_models = [rc_mlp_tfidf,rc_ad_tfidf, rc_nb_tfidf, rc_xgb_tfidf]

plt.figure(figsize=(14,4))

sns.barplot(x = tfidf_models, y= rc_tfidf_models)

tfidf_models = ['MLP Classifier','AdaBoost Classifier', 'MultinomialNB', "XGBoost Classifier"]

acc_tfidf_models = [acc_mlp_tfidf,acc_ad_tfidf, acc_nb_tfidf, acc_xgb_tfidf]

plt.figure(figsize=(14,4))

sns.barplot(x = tfidf_models, y= acc_tfidf_models)

```

## **## ROC Curve Classifiers of cv**

```

Print ('MLP Score: ', roc_auc_score(y_test, mlp_cv_predict))

Print ('AdaBoost Score: ', roc_auc_score(y_test, ad_cv_predict))

Print ('MultinomialNB Score: ', roc_auc_score(y_test, nb_cv_predict))

Print ('XG Boost Score: ', roc_auc_score(y_test, xgb_cv_predict))

mlp_fpr, mlp_tpr, threshold = roc_curve(y_test,mlp_cv_predict)

ad_fpr, ad_tpr, threshold = roc_curve(y_test,ad_cv_predict)

nb_fpr, nb_tpr, threshold = roc_curve(y_test, nb_cv_predict)

xgb_fpr, xgb_tpr, thresold = roc_curve(y_test, xgb_cv_predict)

def graph_roc_curve_multiple(mlp_fpr, mlp_tpr,ad_fpr, ad_tpr, nb_fpr, nb_tpr,xgb_fpr,
xgb_tpr):

    plt.figure(figsize=(8,6))

    plt.title('ROC Curve \n Classifiers', fontsize=18)

    plt.plot(mlp_fpr, mlp_tpr, label='MLP Classifier')

    plt.plot(ad_fpr, ad_tpr, label='AdaBoost Classifier')

    plt.plot(nb_fpr, nb_tpr, label='MultinomialNB')

    plt.plot(xgb_fpr, xgb_tpr, label='XGBoost Classifier')

    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(mlp_fpr, mlp_tpr, ad_fpr, ad_tpr, nb_fpr, nb_tpr, xgb_fpr, xgb_tpr)

plt.show()

```

## ## ROC Curve Classifiers of Tf-Idf

```

Print ('MLP Score: ', roc_auc_score(y_test, mlp_tfidf_predict))

print('AdaBoost Score: ', roc_auc_score(y_test, ad_tfidf_predict))

print('MultinomialNB Score: ', roc_auc_score(y_test, nb_tfidf_predict))

print('XG Boost Score: ', roc_auc_score(y_test, xgb_tfidf_predict))

mlp_fpr, mlp_tpr, threshold = roc_curve(y_test, mlp_tfidf_predict)

ad_fpr, ad_tpr, threshold = roc_curve(y_test, ad_tfidf_predict)

nb_fpr, nb_tpr, threshold = roc_curve(y_test, nb_tfidf_predict)

xgb_fpr, xgb_tpr, threshold = roc_curve(y_test, xgb_tfidf_predict)

def graph_roc_curve_multiple(mlp_fpr, mlp_tpr, ad_fpr, ad_tpr, nb_fpr, nb_tpr, xgb_fpr,
xgb_tpr):

    plt.figure(figsize=(8,6))

    plt.title('ROC Curve \n Classifiers', fontsize=18)

    plt.plot(mlp_fpr, mlp_tpr, label='MLP Classifier')

    plt.plot(ad_fpr, ad_tpr, label='AdaBoost Classifier')

    plt.plot(nb_fpr, nb_tpr, label='MultinomialNB')

    plt.plot(xgb_fpr, xgb_tpr, label='XGBoost Classifier')

```



```

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(mlp_fpr, mlp_tpr, ad_fpr, ad_tpr, nb_fpr, nb_tpr, xgb_fpr, xgb_tpr)

plt.show()

file_name = 'final_tweets.csv'

df.to_csv(file_name, index=False)

```

## Deployment

### App.py

```

from flask import Flask, render_template, url_for, request
import pandas as pd
import pickle
from sklearn.feature_extraction.text import CountVectorizer

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['POST'])
def predict():
    df= pd.read_csv("Data/train_tweets.csv")

```

```

df_data = df[["tweet","label"]]
# Features and Labels
df_x = df_data['tweet']
df_y = df_data.label
# Extract Feature With CountVectorizer
corpus = df_x
cv = CountVectorizer()
#X = cv.fit_transform(corpus) # Fit the Data
X = cv.fit_transform(df['tweet'].values.astype('U'))
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, df_y, test_size=0.20, random_state=42)
#MLPClassifier
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(random_state=42)
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
#Alternative Usage of Saved Model
# ytb_model = open("MLP_model.pkl","rb")
# clf = joblib.load(ytb_model)

if request.method == 'POST':
    comment = request.form['comment']
    data = [comment]
    vect = cv.transform(data).toarray()
    my_prediction = clf.predict(vect)
return render_template('result.html',prediction = my_prediction)

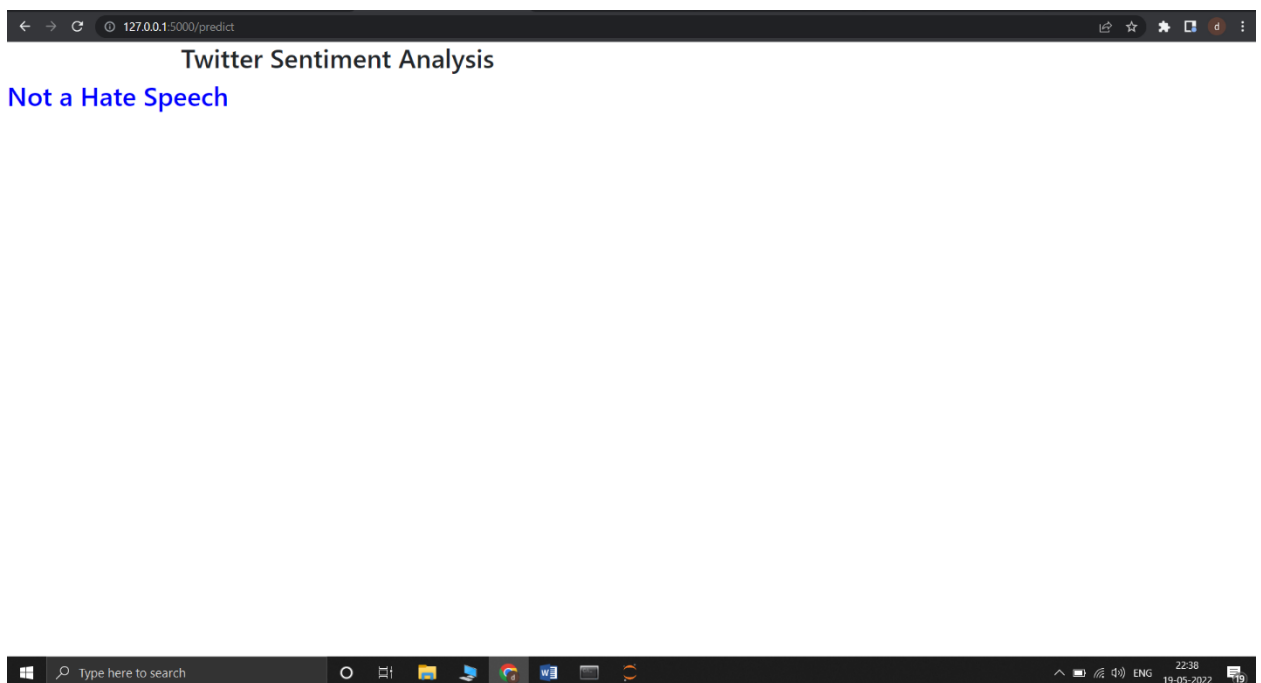
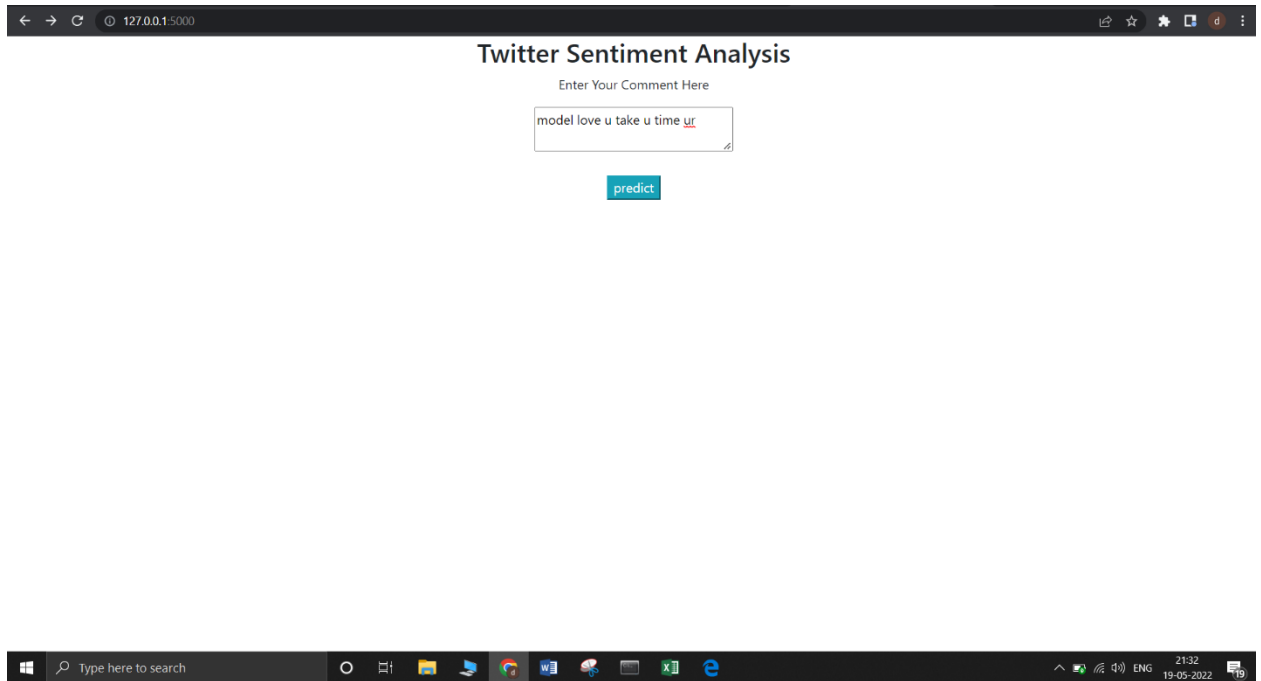
if __name__ == '__main__':
    app.run(debug=True)

```

# APPENDIX-C

## OUTPUT SCREENSHOTS

### DETECTING NOT HATE SPEECH



# DETECTING NOT HATE SPEECH

