FLIP ROBO

NAME OF THE PROJECT

MALIGNANT COMMENT CLASSIFICATION

Submitted by:

V TARAK RAM SAI

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped you and guided you in completion of the project.

INTRODUCTION

- Business Problem Framing

  Sentiment Analysis is the task of analysing people's opinions in textual data (e.g., product reviews, movie reviews, or tweets), and extracting their polarity and viewpoint. The task can be cast as either a binary or a multi-class problem. Binary sentiment analysis classifies texts into positive and negative classes, while multi-class sentiment analysis classifies texts into fine-grained labels or multi-level intensities.

- Conceptual Background of the Domain Problem

  The goal of this project is to use deep learning techniques to identify the level toxicity of a comment which could be used to help deter users from posting potentially hurtful comments, engage in more sophisticated arguments and make internet a safer place. Our dataset consists of 6 labels namely highly malignant, malignant, abuse, threat, loathe and rude. This project aims to implement various Machine Learning algorithms and deep learning algorithms like Multilayer perceptron(MLP), Long Short Term Memory Networks, Multinomial Naïve Bayes, Logistic Regression, Random Forest Classifier, Linear SVC and Adaptive Boosting.

- Review of Literature

  This story demonstrates common **Natural Language Processing** tasks performed specifically to understand the relationship between low ratings and their corresponding comments. In the last years, interest in sentiment analysis in machine learning engineering domain has risen significantly. There are studies that analyse user's satisfaction and developers' emotions automatically by applying sentiment analysis.

- Motivation for the Problem Undertaken
  Machine learning-based systems are growing in popularity in research applications in most disciplines. Considerable decision-making knowledge from data has been acquired in the broad area of machine learning, in which decision-making tree-based ensemble techniques are recognized for supervised classification problems. Thus, classification is an essential form of data analysis in data mining that formulates models while describing significant data classes. Accordingly, such models estimate categorical class labels, which can provide users with an enhanced understanding of the data at large resulted in significant advancements in classification accuracy. Motivated by the preceding literature, we evaluated a large number of machine learning algorithms in our work on credit risk in microlending. A set of algorithms that performed well in numerical experiments with real data is explained in more details below.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem
  Any machine learning model should follow the below steps while dealing a business problem. They are:
  i.) Business Understanding: The first step is to comprehend the research's background, the problem description, and how the proposed project will achieve the goals.
  ii.) **Data Understanding:** The second stage requires collection of data listed in the project resources. This involves in determining the data requirements and exploring key data attributes.
  iii.) **Data Preparation:** The third stage involves data cleaning and should the handle the missing values in the data.
  iv.) **Modelling:** This involves determining the modelling technique and testing the design.

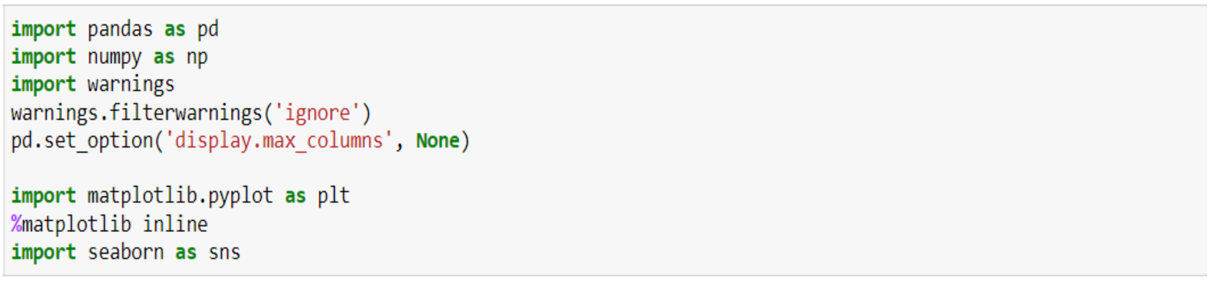
v.) **Evaluation:** Here, we should evaluate the achieved results and should determine the performance of the model with best accuracy.
vi.) **Deployment:** The last stage is implementation of the model.

- Data Sources and their formats

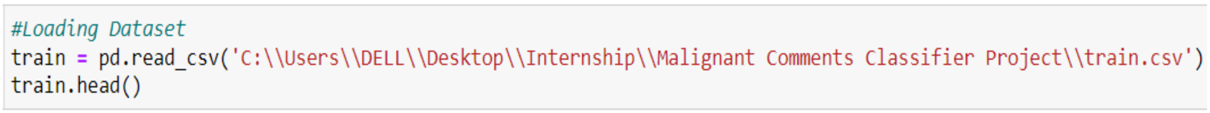
Malignant: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
Highly Malignant: It denotes comments that are highly malignant and hurtful.
Rude: It denotes comments that are very rude and offensive.
Threat: It contains indication of the comments that are giving any threat to someone.
Abuse: It is for comments that are abusive in nature.
Loathe: It describes the comments which are hateful and loathing in nature.
ID: It includes unique Ids associated with each comment text given.
Comment text: This column contains the comments extracted from various social media platforms.

- Data Pre-processing

It entails converting raw data into comprehensible format that a machine learning model can understand. The data pre-processing involves data cleaning which involves handling missing values, transformation of data.

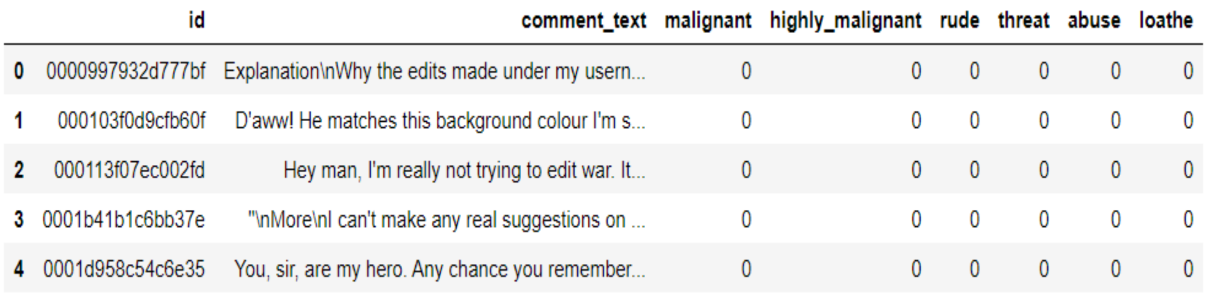


```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```



```
#Loading Dataset
train = pd.read_csv('C:\\Users\\DELL\\Desktop\\Internship\\Malignant Comments Classifier Project\\train.csv')
train.head()
```

The head of the primary table is as follows:

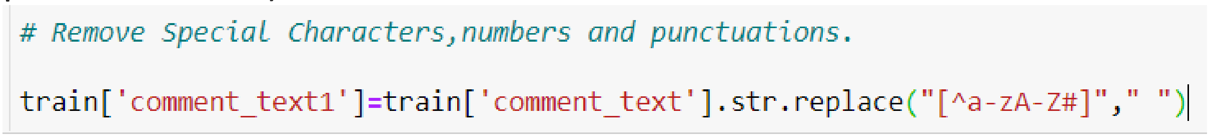


	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Preparing the data

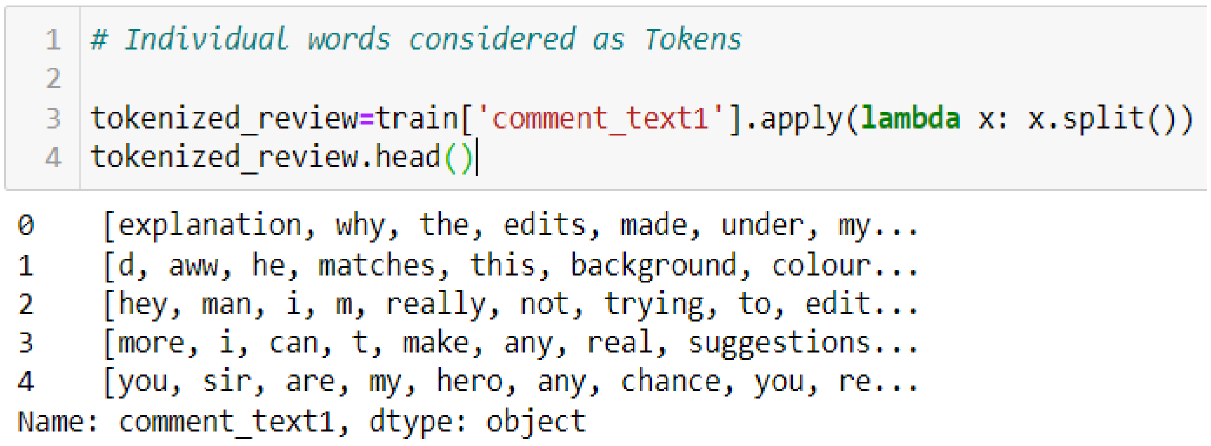
Raw text data are messy to work with and pre-processing steps are needed before training the computer on them. There are many different ways and order to prepare the text data. In this example, I perform the following tasks before loading the data into a DataFrame:

- **Text Normalization** — remove punctuations using Regular Expressions (Regex can be used to find basic and complex patterns in text).



```
# Remove Special Characters,numbers and punctuations.

train['comment_text1']=train['comment_text'].str.replace("[^a-zA-Z#]"," ")
```

- **Tokenize —** tokenization is the process of splitting text into smaller pieces called tokens.



```
# Individual words considered as Tokens

tokenized_review=train['comment_text1'].apply(lambda x: x.split())
tokenized_review.head()
```

```
0    [explanation, why, the, edits, made, under, my...
1    [d, aww, he, matches, this, background, colour...
2    [hey, man, i, m, really, not, trying, to, edit...
3    [more, i, can, t, make, any, real, suggestions...
4    [you, sir, are, my, hero, any, chance, you, re...
Name: comment_text1, dtype: object
```

- **Remove stop words** — stop words are words that are usually filtered out before pre-processing of the text. They are generally the most common words in a language that search engines have been programmed to ignore such as personal pronouns (I, you, he, she, it, etc.).

```
# Making a Stop Words list
import nltk
nltk.download('stopwords')

stopwords= nltk.corpus.stopwords.words('english')
```

```
# Removal of Stop Words

def rem_stop(tokenized_txt):
    txt_clean=[word for word in tokenized_txt if word not in stopwords]
    return txt_clean
```

```
tokenized_review=tokenized_review.apply(lambda x: rem_stop(x))
tokenized_review.head()
```

```
0    [explanation, edits, made, username, hardcore,...
1    [aww, matches, background, colour, seemingly, ...
2    [hey, man, really, trying, edit, war, guy, con...
3    [make, real, suggestions, improvement, wondere...
4                  [sir, hero, chance, remember, page]
Name: comment_text1, dtype: object
```

- **Lemmatization** — stemming is the process of removing a part of a word, or *reducing a word to its stem* or root. This might not necessarily mean we're reducing a word to its dictionary root. We use a few algorithms to decide how to chop a word off.

```
#Performing Lemmatization of words
from nltk.stem import WordNetLemmatizer
def word_lemmatizer(text):
    lem_text = [WordNetLemmatizer().lemmatize(i) for i in text]
    return lem_text
tokenized_review = tokenized_review.apply(lambda x: word_lemmatizer(x))

tokenized_review.head()
```

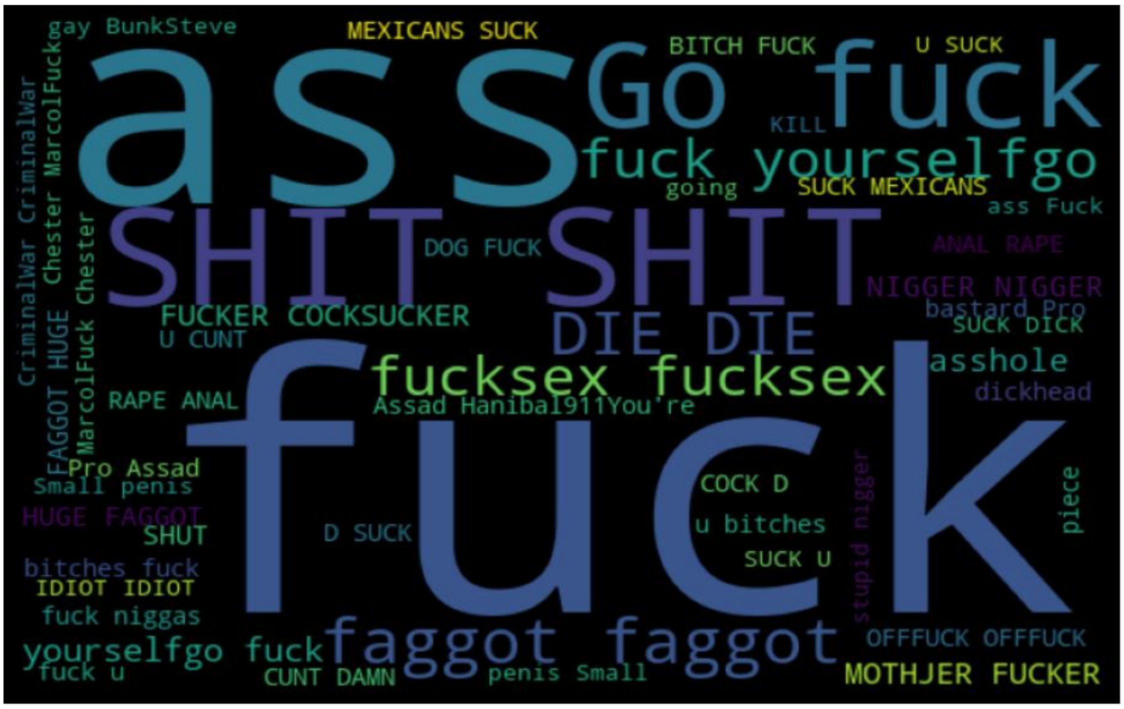
```
0    [explanation, edits, made, username, hardcore,...
1    [aww, match, background, colour, seemingly, st...
2    [hey, man, really, trying, edit, war, guy, con...
3    [make, real, suggestion, improvement, wondered...
4                  [sir, hero, chance, remember, page]
Name: comment_text1, dtype: object
```

- **Vectorize the text** — in order for the computer to understand the text data, they need to be converted into a numerical representation, which is commonly described as **Word Embedding**. The most intuitive form of this process can be done through a form of one-hot encoding known as the bag-of-words and it describes the occurrence of words. In a bag-of-words, the columns represent each of the unique words in the entire corpus of documents, the rows represent each document, sentence, or topic, and the cells capture the number of times each word appears in each row.

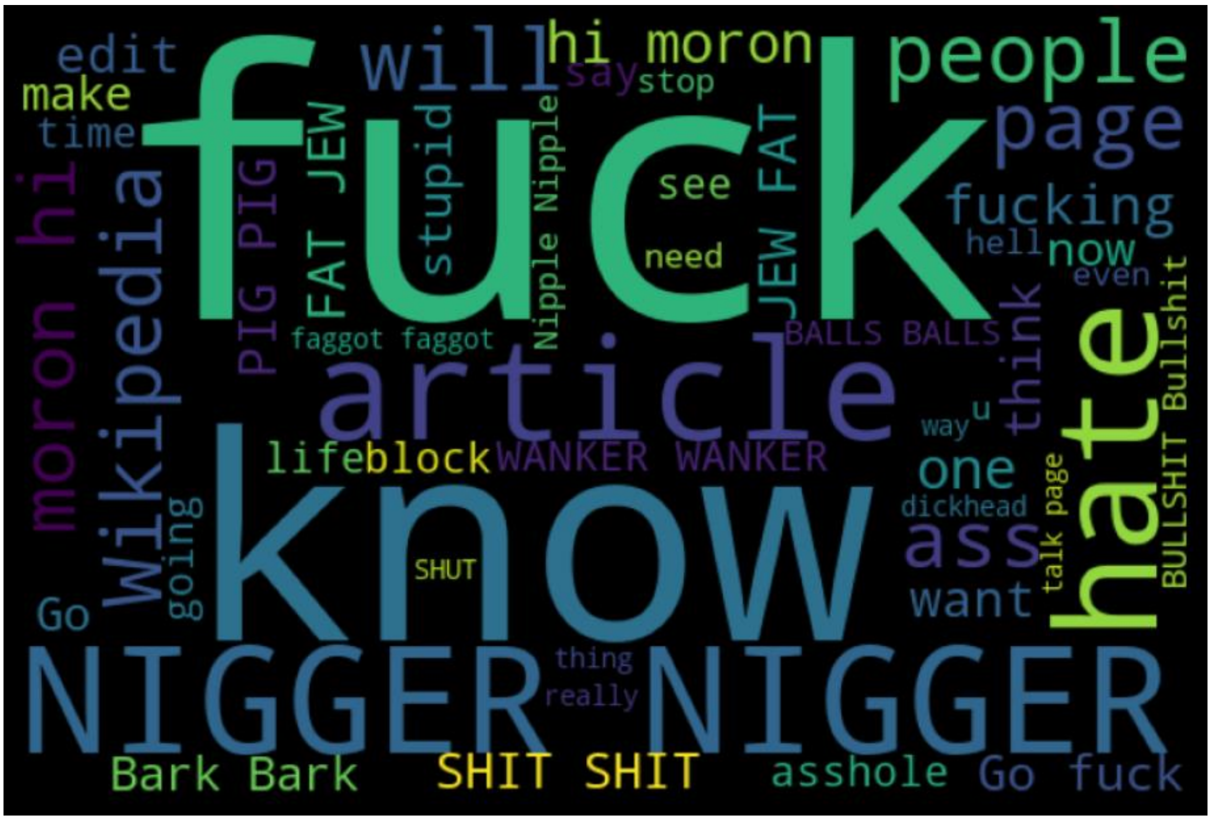
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vec = TfidfVectorizer(max_features = 20000, stop_words='english')
features = tf_vec.fit_transform(train['clean_comment'])
```

- Data Inputs- Logic- Output Relationships

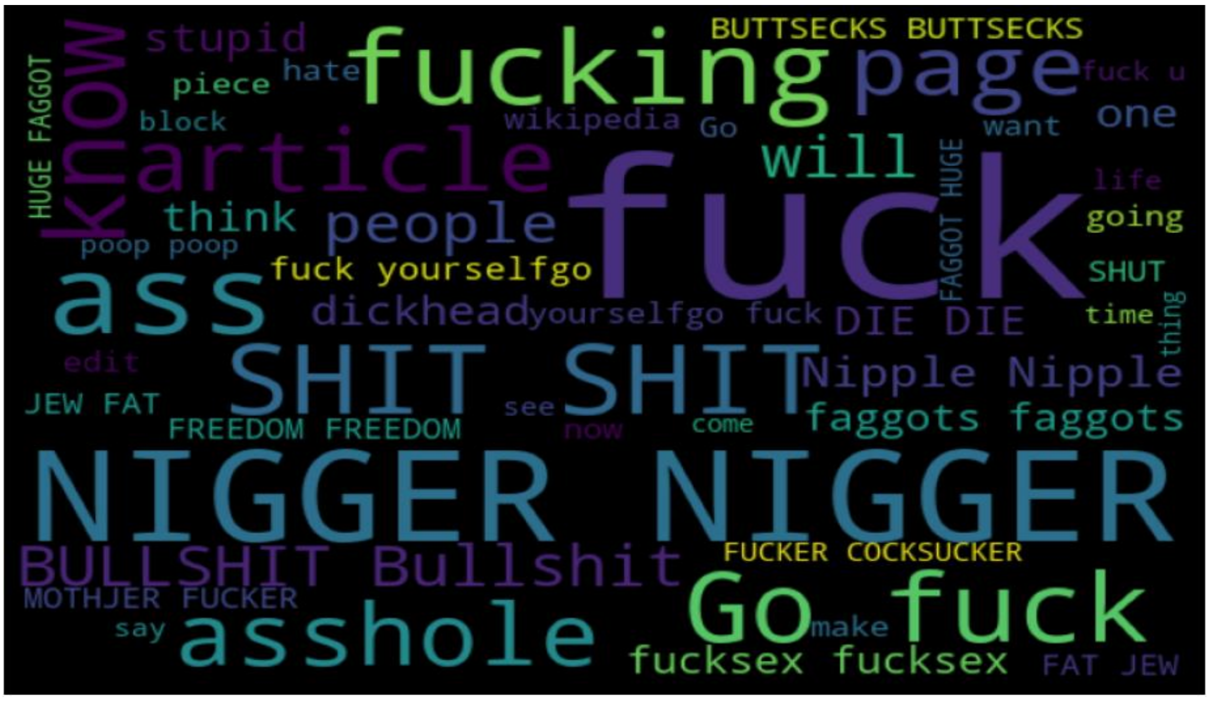
The Word Cloud for the highly malignant comments is as follows-



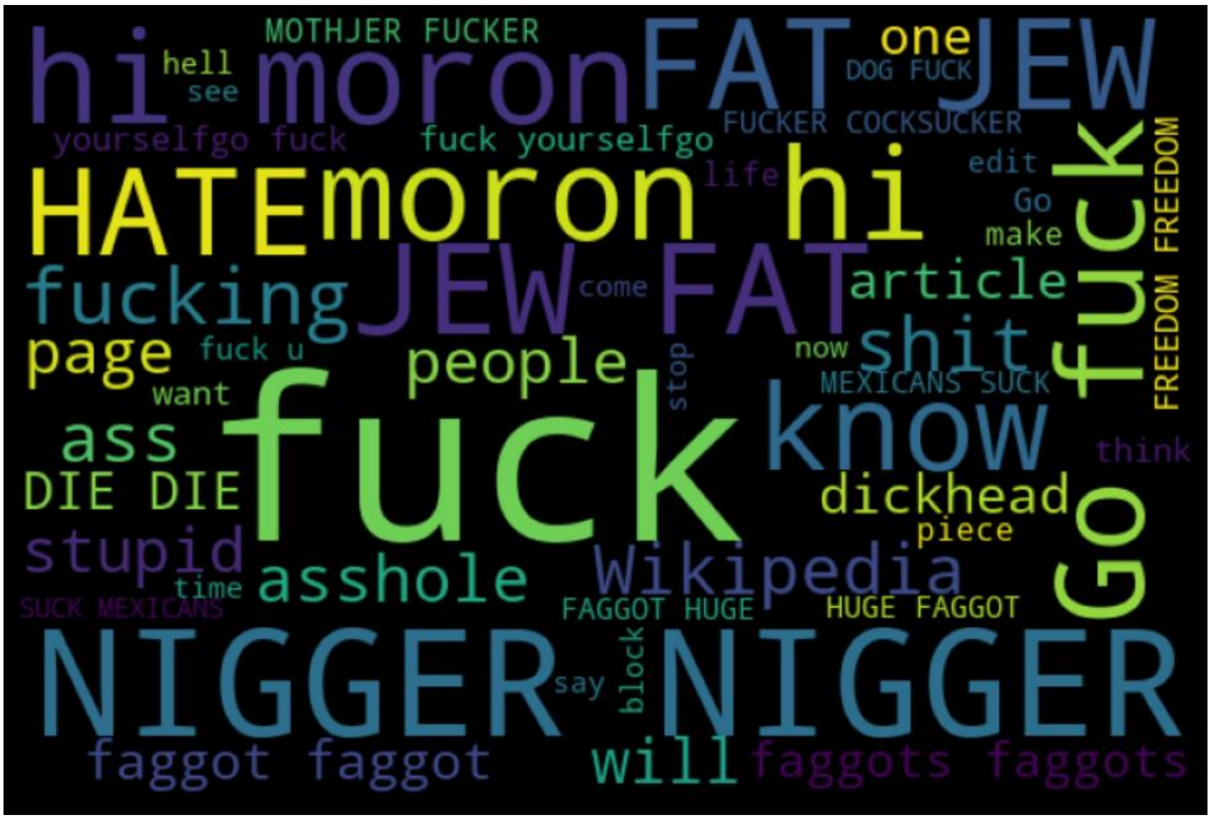
The Word Cloud for the malignant comments is as follows-



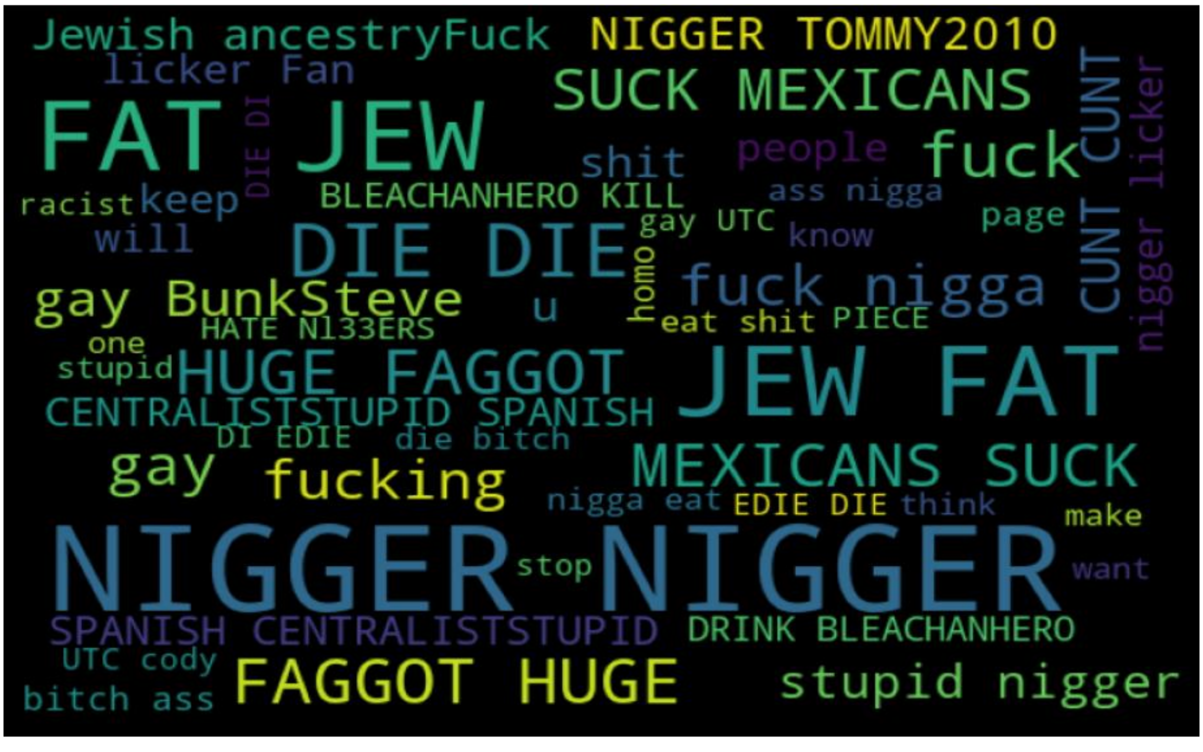
The Word Cloud for the Rude Comments is as follows-



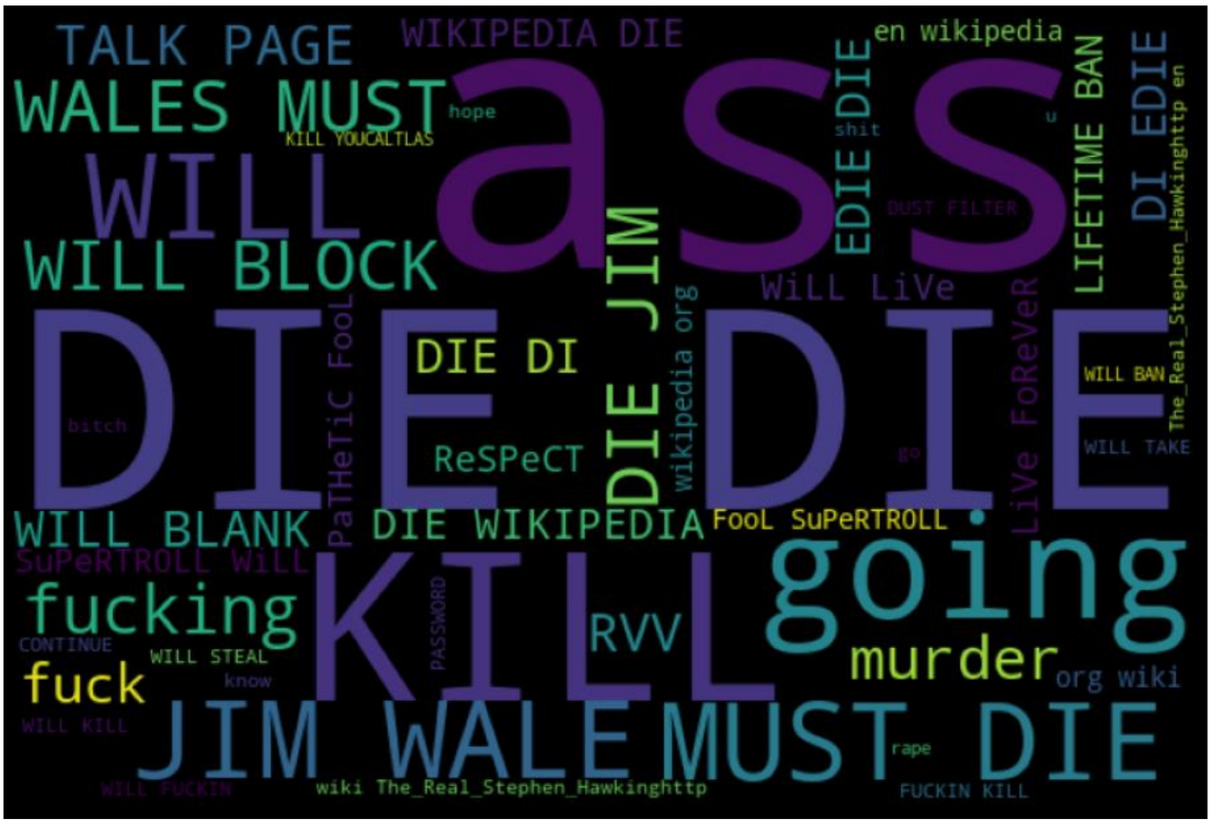
The Word Cloud for the Abuse Comments is as follows-



The Word Cloud for the Loathe Comments is as follows-



The Word Cloud for Threat Comments is as follows-



- Hardware and Software Requirements and Tools Used

The hardware requirements for the project includes a laptop with at least 4GB RAM. This project uses a Jupyter Notebook as a code editor. The Machine Learning models are implemented using python version 3.7 with libraries like numpy, pandas, matplotlib, seaborn and sklearn.

Model/s Development and Evaluation

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=56,test_size=.30)
```

- **Logistic Regression**

  Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

  1 / (1 + e^-value),

  Where e is the base of the natural logarithms (Euler’s number or the EXP () function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modelled is a binary value (0 or 1) rather than a numeric value.

  y = e^ (b0 + b1*x) / (1 + e^ (b0 + b1*x))

  Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

```python
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)

LG.fit(X_train, y_train)
```

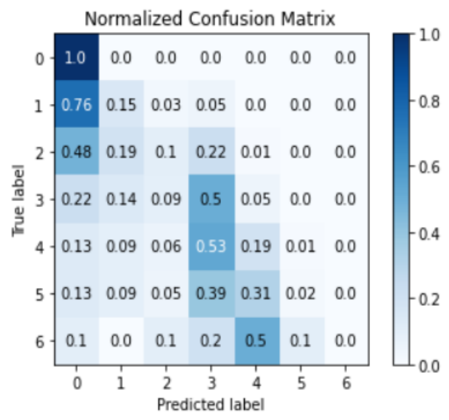
```
LogisticRegression(C=1, max_iter=3000)
```

```python
# Performing Evaluation metrics for our model
predlg = LG.predict(X_test)

print('Accuracy Score for Logistic Regression is :',round(accuracy_score(y_test,predlg),4)*100,'% \n')
print('Confusion Matrix of Logistic Regression : \n',confusion_matrix(y_test,predlg), '\n')
print('Classification Logistic Regression : \n',classification_report(y_test,predlg))
print("Hamming Loss for our Logistic Regression model is : ",hamming_loss(y_test,predlg))
```

```
Accuracy Score for Logistic Regression is : 91.83 %

Confusion Matrix of Logistic Regression :
 [[42831   100     8    11     0     0     0]
 [ 1477   298    66    97     9     0     0]
 [  504   201   101   235    10     0     0]
 [  286   173   118   632    65     1     0]
 [   68    50    30   277    99     3     0]
 [   15    10     6    44    35     2     0]
 [    1     0     1     2     5     1     0]]

Classification Logistic Regression :
               precision    recall  f1-score   support

           0       0.95      1.00      0.97     42950
           1       0.36      0.15      0.21      1947
           2       0.31      0.10      0.15      1051
           3       0.49      0.50      0.49      1275
           4       0.44      0.19      0.26       527
           5       0.29      0.02      0.03       112
           6       0.00      0.00      0.00        10

    accuracy                           0.92     47872
   macro avg       0.40      0.28      0.30     47872
weighted avg       0.89      0.92      0.90     47872

Hamming Loss for our Logistic Regression model is :  0.0816552473262032
```

```python
skplt.metrics.plot_confusion_matrix(y_test, predlg, normalize=True)
```

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



- **Random Forest Classifier**

  The random forest classifier is an ensemble method algorithm of decision trees wherein each tree depends on randomly selected samples trained independently, with a similar distribution for all the trees in the forest (Breiman 2001). Hence, a random forest is a classifier incorporating a collection of tree-structured classifiers that decrease overfitting, resulting in an increase in the overall accuracy (Geurts et al. 2006). As such, random forest's accuracy differs based on the strength of each tree classifier and their dependencies.

```
RF = RandomForestClassifier(max_depth=7)
```

```
RF.fit(X_train,y_train)
```

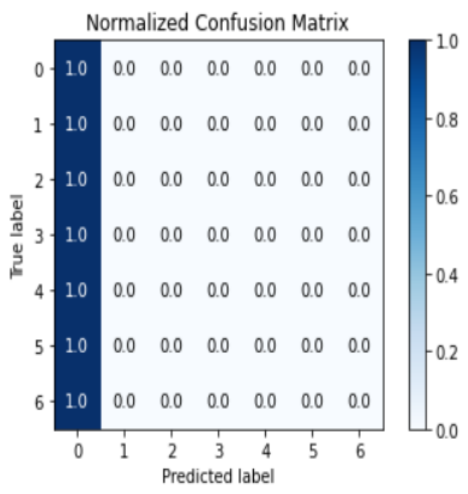
```
RandomForestClassifier(max_depth=7)
```

```
# Performing Evaluation metrics for our model
predrf=RF.predict(X_test)
print('Accuracy Score for Random Forest Classifier is :',round(accuracy_score(y_test,predrf),4)*100,
print('Confusion Matrix of Random Forest Classifier : \n',confusion_matrix(y_test,predrf), '\n')
print('Classification Random Forest Classifier : \n',classification_report(y_test,predrf))
print("Hamming Loss for our Random Forest Classifier model is : ",hamming_loss(y_test,predrf))
```

```
Accuracy Score for Random Forest Classifier is : 89.72 %

Confusion Matrix of Random Forest Classifier :
 [[42950     0     0     0     0     0     0]
 [ 1947     0     0     0     0     0     0]
 [ 1051     0     0     0     0     0     0]
 [ 1275     0     0     0     0     0     0]
 [  527     0     0     0     0     0     0]
 [  112     0     0     0     0     0     0]
 [   10     0     0     0     0     0     0]]

Classification Random Forest Classifier :
               precision    recall  f1-score   support

           0       0.90      1.00      0.95     42950
           1       0.00      0.00      0.00      1947
           2       0.00      0.00      0.00      1051
           3       0.00      0.00      0.00      1275
           4       0.00      0.00      0.00       527
           5       0.00      0.00      0.00       112
           6       0.00      0.00      0.00        10

    accuracy                           0.90     47872
   macro avg       0.13      0.14      0.14     47872
weighted avg       0.80      0.90      0.85     47872

Hamming Loss for our Random Forest Classifier model is :  0.1028158422459893
```

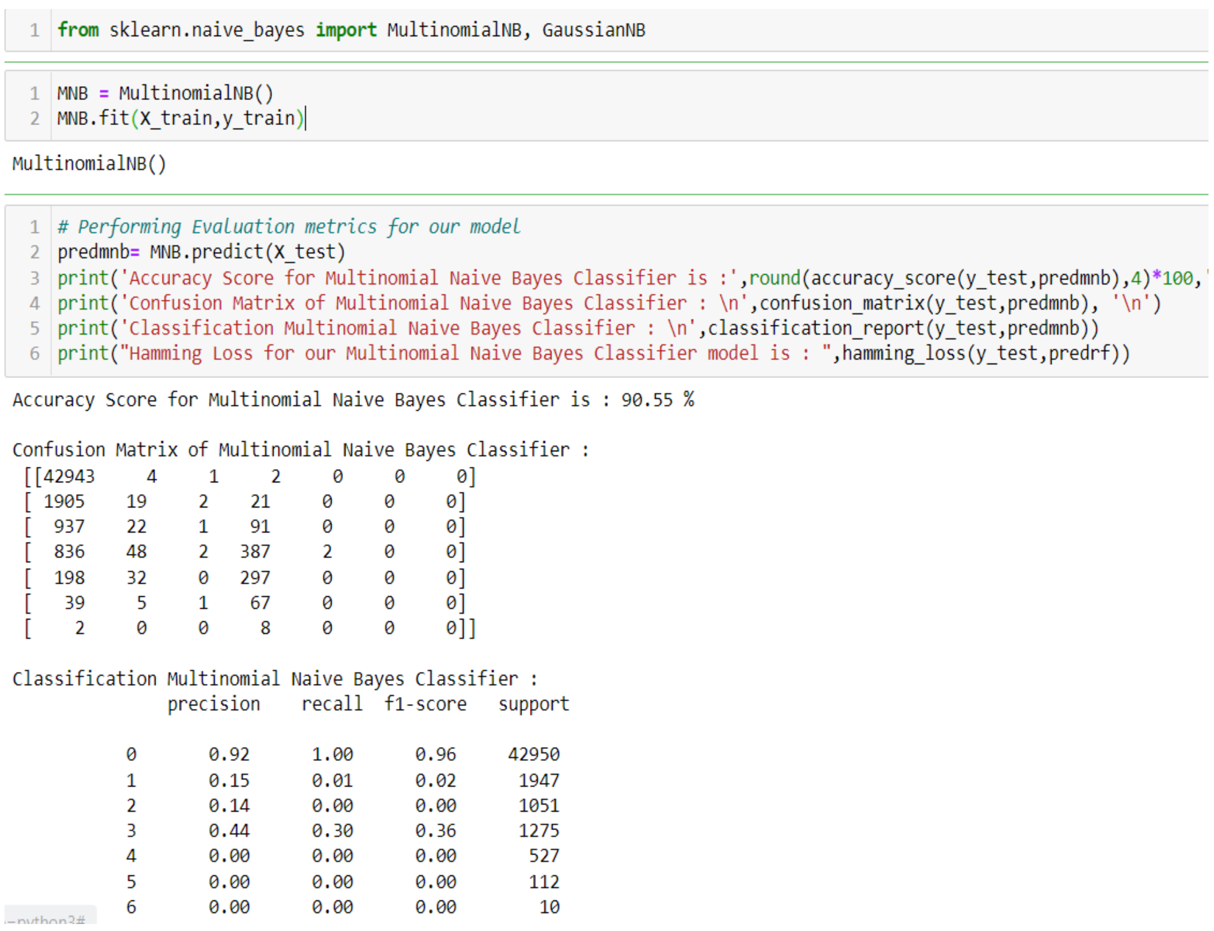
```
#Plotting Confusion Matrix of Logistic Regression model
skplt.metrics.plot_confusion_matrix(y_test, predrf, normalize=True)
```

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



- **Multinomial Naïve Bayes**

  Multinomial Naive Bayes algorithm is a probabilistic learning method that is mostly used in Natural Language Processing (NLP). The algorithm is based on the Bayes theorem and predicts the tag of a text such as a piece of email or newspaper article. It calculates the probability of each tag for a given sample and then gives the tag with the highest probability as output.
  Naive Bayes is a powerful algorithm that is used for text data analysis and with problems with multiple classes. To understand Naive Bayes theorem’s working, it is important to understand the Bayes theorem concept first as it is based on the latter. Bayes theorem, formulated by Thomas Bayes, calculates the probability of an event occurring based on the prior knowledge of conditions related to an event.



```
from sklearn.naive_bayes import MultinomialNB, GaussianNB
```

```
MNB = MultinomialNB()
MNB.fit(X_train,y_train)
```

```
MultinomialNB()
```

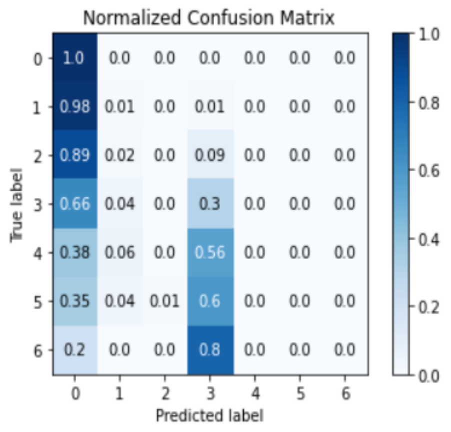
```
# Performing Evaluation metrics for our model
predmnb= MNB.predict(X_test)
print('Accuracy Score for Multinomial Naive Bayes Classifier is :',round(accuracy_score(y_test,predmnb),4)*100,'
print('Confusion Matrix of Multinomial Naive Bayes Classifier : \n',confusion_matrix(y_test,predmnb), '\n')
print('Classification Multinomial Naive Bayes Classifier : \n',classification_report(y_test,predmnb))
print("Hamming Loss for our Multinomial Naive Bayes Classifier model is : ",hamming_loss(y_test,predrf))
```

```
Accuracy Score for Multinomial Naive Bayes Classifier is : 90.55 %

Confusion Matrix of Multinomial Naive Bayes Classifier :
 [[42943     4     1     2     0     0     0]
 [ 1905    19     2    21     0     0     0]
 [  937    22     1    91     0     0     0]
 [  836    48     2   387     2     0     0]
 [  198    32     0   297     0     0     0]
 [   39     5     1    67     0     0     0]
 [    2     0     0     8     0     0     0]]

Classification Multinomial Naive Bayes Classifier :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96     42950
           1       0.15      0.01      0.02      1947
           2       0.14      0.00      0.00      1051
           3       0.44      0.30      0.36      1275
           4       0.00      0.00      0.00       527
           5       0.00      0.00      0.00       112
           6       0.00      0.00      0.00        10
```

```
    accuracy                           0.91     47872
   macro avg       0.24      0.19      0.19     47872
weighted avg       0.84      0.91      0.87     47872

Hamming Loss for our Multinomial Naive Bayes Classifier model is :  0.1028158422459893
```

```
#Plotting Confusion Matrix of Logistic Regression model
skplt.metrics.plot_confusion_matrix(y_test, predmnb, normalize=True)
```

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



- ## Ada Boost Classifier

  Adaptive boosting (AdaBoost) is an ensemble algorithm incorporated by Freund and Schapire (1997), which trains and deploys trees in time series. Since then, it evolved as a popular boosting technique introduced in various research disciplines. It merges a set of weak classifiers to build and boost a robust classifier that will improve the decision tree's performance and improve accuracy.

```
from sklearn.ensemble import AdaBoostClassifier
```

```
#Instantiating SVC model and training it

ada = AdaBoostClassifier()
ada.fit(X_train,y_train)
```

```
AdaBoostClassifier()
```

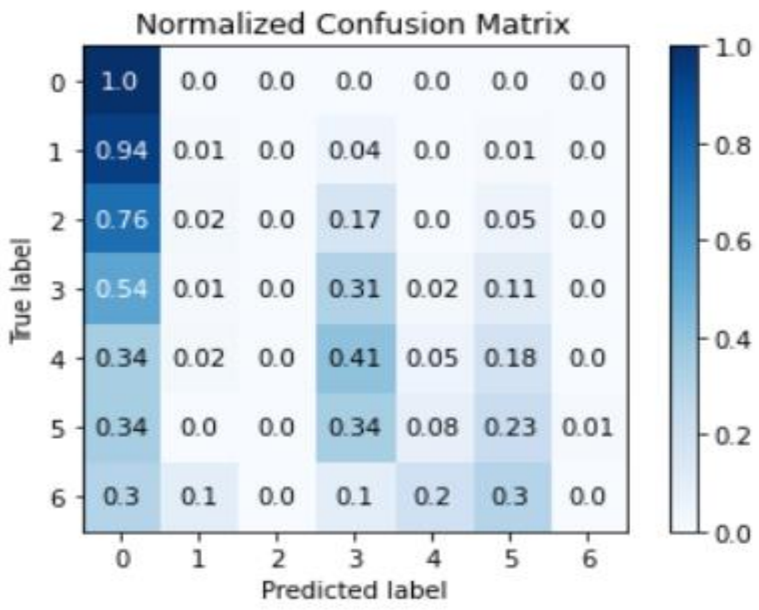
```
# Performing Evaluation metrics for our model
predada= ada.predict(X_test)
print('Accuracy Score for Adaptive Boost Classifier is :',round(accuracy_score(y_test,predada),4)*100,'
print('Confusion Matrix of Adaptive Boost Classifier : \n',confusion_matrix(y_test,predada), '\n')
print('Classification Adaptive Boost Classifier : \n',classification_report(y_test,predada))
print("Hamming Loss for our Adaptive Boost Classifier is : ",hamming_loss(y_test,predada))
```

```
Accuracy Score for Adaptive Boost Classifier is : 90.46 %
```

```
Confusion Matrix of Adaptive Boost Classifier :
 [[42824    69     0    53     3     1     0]
 [ 1827    28     0    80     0    12     0]
 [  796    18     0   178     4    55     0]
 [  687    14     0   401    28   143     2]
 [  178    10     1   215    24    97     2]
 [   38     0     0    38     9    26     1]
 [    3     1     0     1     2     3     0]]

Classification Adaptive Boost Classifier :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96     42950
           1       0.20      0.01      0.03      1947
           2       0.00      0.00      0.00      1051
           3       0.42      0.31      0.36      1275
           4       0.34      0.05      0.08       527
           5       0.08      0.23      0.12       112
           6       0.00      0.00      0.00        10

    accuracy                           0.90     47872
   macro avg       0.28      0.23      0.22     47872
weighted avg       0.85      0.90      0.87     47872

Hamming Loss for our Adaptive Boost Classifier is :  0.09544201203208556
```

```python
#Plotting Confusion Matrix of AdaBoost model
skplt.metrics.plot_confusion_matrix(y_test, predada, normalize=True)
```

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



Cross Validation

Cross Validation is a **very useful technique for assessing the effectiveness of your model**, particularly in cases where you need to mitigate overfitting.

```
CVscore_LR = cross_val_score(LG,X,y,cv = 10 )
print("Cross validation score of Logistic Regression is :", round(CVscore_LR.mean(),4)*100,'%')

CVscore_RF = cross_val_score(RF,X,y,cv = 10 )
print("Cross validation score of Random Forest Classifier :", round(CVscore_RF.mean(),4)*100,'%')

CVscore_adaboost = cross_val_score(ada,X,y,cv = 10 )
print("Cross validation score of AdaBoost Classifier is :",round(CVscore_adaboost.mean(),4)*100,'%')

CVscore_mnb = cross_val_score(MNB,X,y,cv = 10 )
print("Cross validation score of MNB Classifier is :",round(CVscore_mnb.mean(),4)*100,'%')
```

```
Cross validation score of Logistic Regression is : 91.99000000000001 %
Cross validation score of Random Forest Classifier : 89.83 %
Cross validation score of AdaBoost Classifier is : 90.83 %
Cross validation score of MNB Classifier is : 90.81 %
```

We can choose MNB as our model since its accuracy and cv score are almost same.

Hyper Parameter Tuning

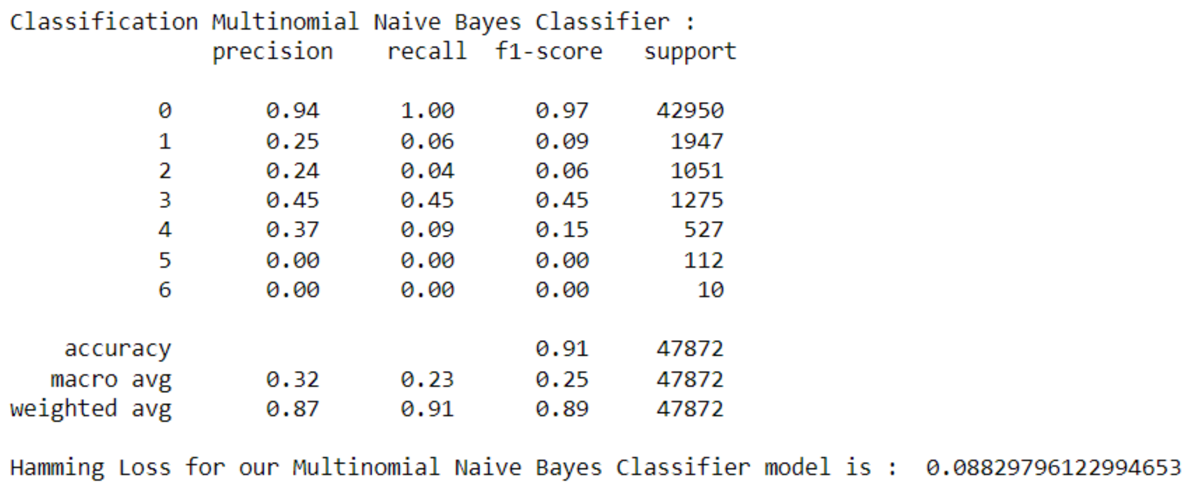
In machine learning, hyper parameter optimization or tuning is **the problem of choosing a set of optimal hyper parameters for a learning algorithm**. A hyper parameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

```
# Multinomial Naive Bayes

parameters = {'fit_prior':[True,False], 'alpha':[0.0001,0.001,0.01,0.1,1,10,100]}
MNB = MultinomialNB()
clf = GridSearchCV(MNB,parameters)
clf.fit(X_train,y_train)

print(clf.best_params_)
```

```
{'alpha': 0.1, 'fit_prior': True}
```

```
fin_MNB = MultinomialNB(alpha=0.1,fit_prior=True)
fin_MNB.fit(X_train,y_train)
fpredmnb= fin_MNB.predict(X_test)
print('Accuracy Score for Multinomial Naive Bayes Classifier is :',round(accuracy_score(y_test,fpredmnb),4)*100,'% \n')
print('Confusion Matrix of Multinomial Naive Bayes Classifier : \n',confusion_matrix(y_test,fpredmnb), '\n')
print('Classification Multinomial Naive Bayes Classifier : \n',classification_report(y_test,fpredmnb))
print("Hamming Loss for our Multinomial Naive Bayes Classifier model is : ",hamming_loss(y_test,fpredmnb))
```

```
Accuracy Score for Multinomial Naive Bayes Classifier is : 91.17 %

Confusion Matrix of Multinomial Naive Bayes Classifier :
 [[42867    49     7    24     2     1     0]
 [ 1713   111    24    95     4     0     0]
 [  714    85    39   209     4     0     0]
 [  447   143    62   578    45     0     0]
 [   85    37    26   329    50     0     0]
 [   13    13     3    57    26     0     0]
 [    1     0     0     6     3     0     0]]
```



```
Classification Multinomial Naive Bayes Classifier :
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     42950
           1       0.25      0.06      0.09      1947
           2       0.24      0.04      0.06      1051
           3       0.45      0.45      0.45      1275
           4       0.37      0.09      0.15       527
           5       0.00      0.00      0.00       112
           6       0.00      0.00      0.00        10

    accuracy                           0.91     47872
   macro avg       0.32      0.23      0.25     47872
weighted avg       0.87      0.91      0.89     47872

Hamming Loss for our Multinomial Naive Bayes Classifier model is :  0.08829796122994653
```

CONCLUSION

- This research evaluated the malicious comment text classification using machine learning techniques. Using real data, we compared the various machine learning algorithms' accuracy by performing detailed experimental analysis while classifying the text into 7 categories.
- Generally, **Multinomial Naïve Bayes** Classification machine learning algorithm has shown a better performance with our real-life data than others, and the most performing models are all ensemble classifiers. It was found that the Random Forest classifier generated the least accurate predictions.