FLIP ROBO

MALAIGNANT COMMENTS CLASSIFICATION

<u>Submitted by:</u>

Praveen Pandey

ACKNOWLEDGMENT

I would like to express my gratitude to my guide Shwetank Mishra (SME, Flip Robo) for his constant guidance, continuous encouragement and unconditional help towards the development of the project. It was he who helped me whenever I got stuck somewhere in between. The project would have not been completed without his support and confidence he showed towards me.

Lastly, I would l like to thank all those who helped me directly or indirectly toward the successful completion of the project.

INTRODUCTION

Business Problem Framing

The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users. Although researchers have found that hate is a problem across multiple platforms, there is a lack of models for online hate detection.

Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying.

Conceptual Background of the Domain Problem

Online platforms and social media become the place where people share the thoughts freely without any partiality and overcoming all the race people share their thoughts and ideas among the crowd.

Social media is a computer-based technology that facilitates the sharing of ideas, thoughts, and information through the building of virtual networks and communities. By design, social media is Internet-based and gives users quick electronic communication of content. Content includes personal information, documents, videos, and photos. Users engage with social media via a computer, tablet, or smartphone via web-based software or applications.

While social media is ubiquitous in America and Europe, Asian countries like India lead the list of social media usage. More than 3.8 billion people use social media.

In this huge online platform or an online community there are some people or some motivated mob wilfully bully others to make them not to share their thought in rightful way. They bully others in a foul language which among the civilized society is seen as ignominy. And when innocent individuals are being bullied by these mob these individuals are going silent without speaking anything. So, ideally the motive of this disgraceful mob is achieved.

To solve this problem, we are now building a model that identifies all the foul language and foul words, using which the online platforms like social media principally stops these mob using the foul language in an online community or even block them or block them from using this foul language.

Review of Literature

The purpose of the literature review is to:

1. Identify the foul words or foul statements that are being used.
2. Stop the people from using these foul languages in online public forum.

To solve this problem, we are now building a model using our machine language technique that identifies all the foul language and foul words, using which the online platforms like social media principally stops these mob using the foul language in an online community or even block them or block them from using this foul language.

I have used 5 different Classification algorithms and shortlisted the best on basis on the metrics of performance and I have chosen one algorithm and build a model in that algorithm.

Motivation for the Problem Undertaken

I am doing this for practice, to get more hands-on data exploration, Feature extraction and Model building.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

I start analysis on this project in importing the data set and simple play around with the data and identifying the characteristics of each column.

```
df.dtypes
```

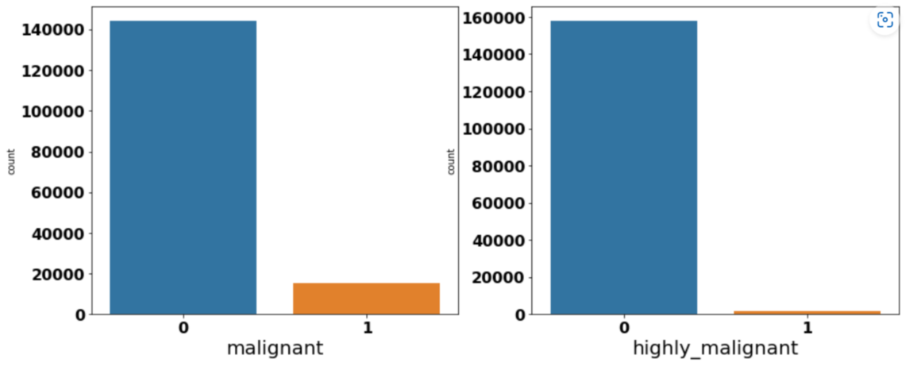
```
id                   object
comment_text         object
malignant             int64
highly_malignant      int64
rude                  int64
threat                int64
abuse                 int64
loathe                int64
dtype: object
```

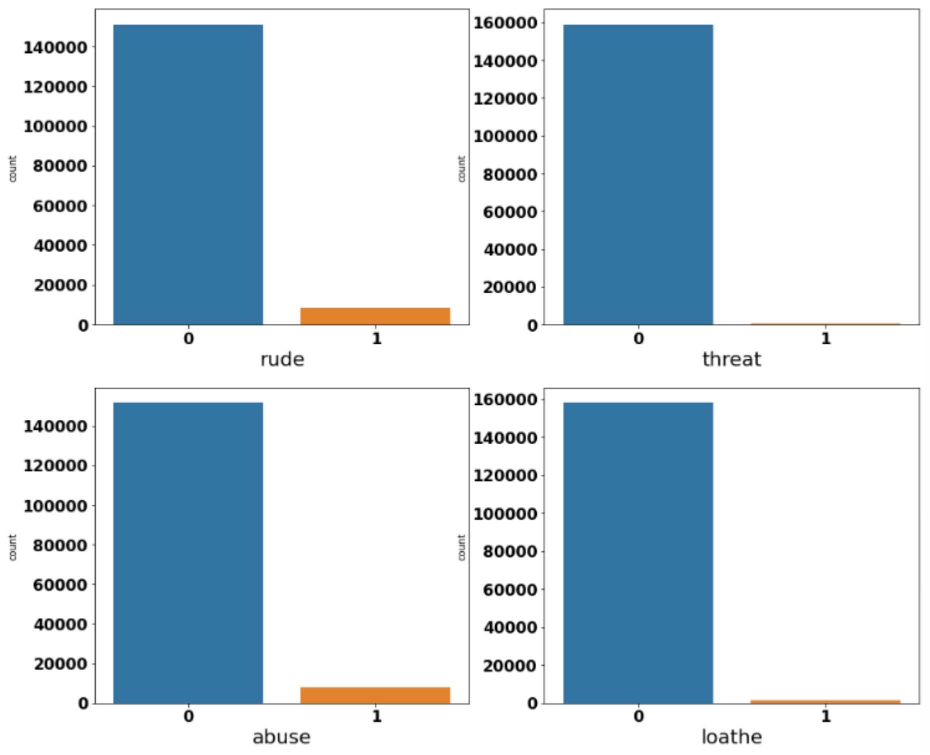
In the first stoke of analysis I understood that there are 8 columns in total in which 6 are numerical column with binary data 0's and 1's and 'id' data which has all unique values "connect text" have string values.

Since 'id' have all unique values, it won't be helpful in analysis to I have dropped id column.

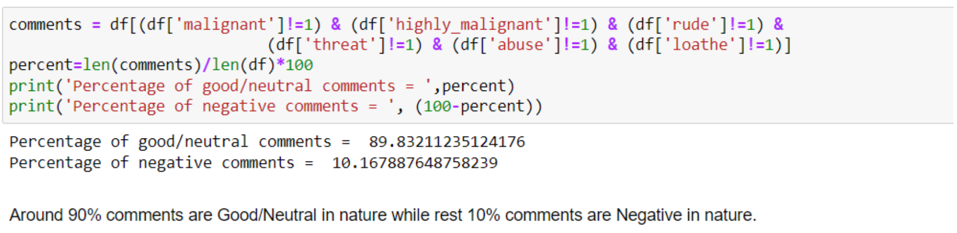
```
df.drop('id',axis=1,inplace=True)
dft.drop('id',axis=1,inplace=True)
```

Post dropping 'id' I tried to understand data from 'malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe columns by plotting them in count plot.





we can see that there only minimum number of columns in 'malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe' and remaining all in 0.

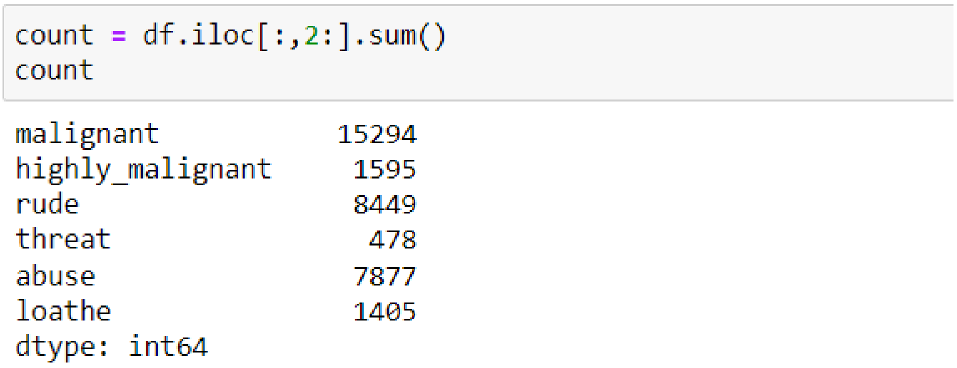


```
comments = df[(df['malignant']!=1) & (df['highly_malignant']!=1) & (df['rude']!=1) &
                        (df['threat']!=1) & (df['abuse']!=1) & (df['loathe']!=1)]
percent=len(comments)/len(df)*100
print('Percentage of good/neutral comments = ',percent)
print('Percentage of negative comments = ', (100-percent))
```

```
Percentage of good/neutral comments =  89.83211235124176
Percentage of negative comments =  10.167887648758239
```

Around 90% comments are Good/Neutral in nature while rest 10% comments are Negative in nature.

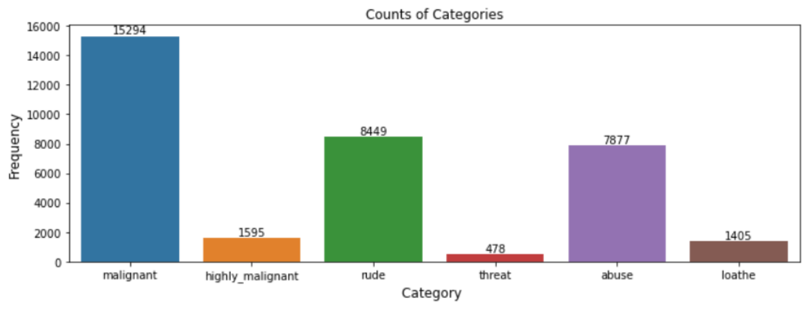
So, only 10% of the data is getting classified as malignant comments data is unbalanced.



```
count = df.iloc[:,2:].sum()
count
```

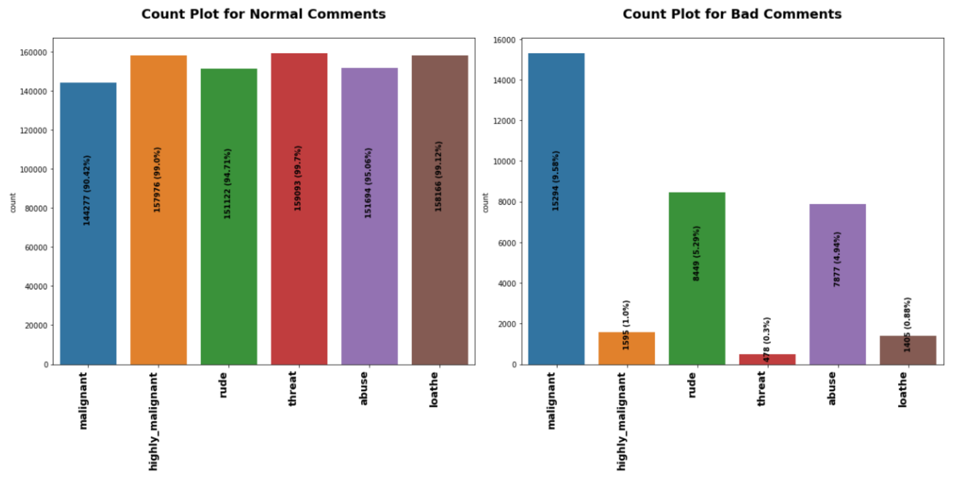
```
malignant           15294
highly_malignant     1595
rude                 8449
threat                478
abuse                7877
loathe               1405
dtype: int64
```

```python
plt.figure(figsize=(12,4))
ax = sns.barplot(count.index, count.values)
plt.title("Counts of Categories")
plt.ylabel('Frequency', fontsize=12)
plt.xlabel('Category ', fontsize=12)
rects = ax.patches
labels = count.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')
plt.show()
```

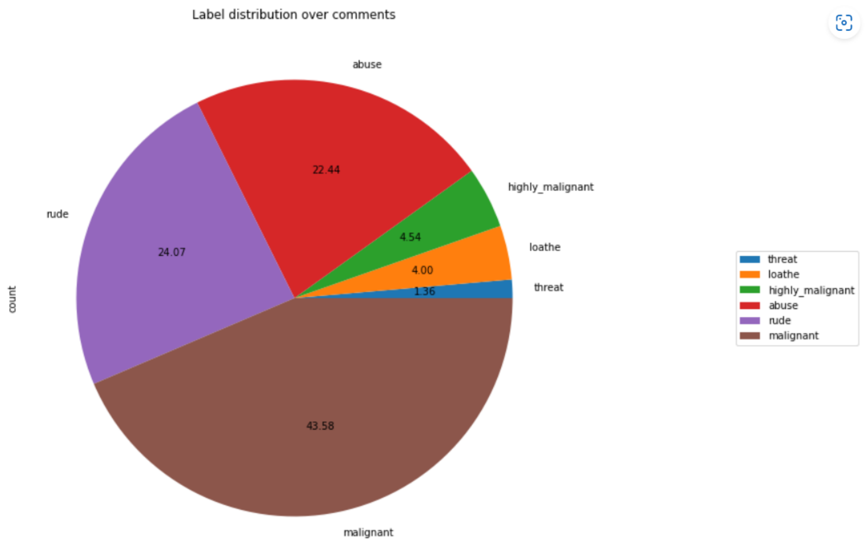


We can see malignant and rude are high categorised sentences in the data.

```python
fig, ax = plt.subplots(1,2,figsize=(18,9))

for i in range(2):
    sns.countplot(data=df[df.columns[2:]][df[df.columns[2:]]==i], ax=ax[i])
    if i == 0:
        ax[i].set_title("Count Plot for Normal Comments\n", fontsize=18, fontweight='bold')
    else:
        ax[i].set_title("Count Plot for Bad Comments\n", fontsize=18, fontweight='bold')

    ax[i].set_xticklabels(df.columns[2:], rotation=90, ha="right", fontsize=14, fontweight='bold')
    p=0
    for prop in ax[i].patches:
        count = prop.get_height()
        s = f"{count} ({round(count*100/len(df),2)}%)"
        ax[i].text(p,count/2,s,rotation=90, ha="center", fontweight="bold")
        p += 1
plt.tight_layout()
plt.show()
```

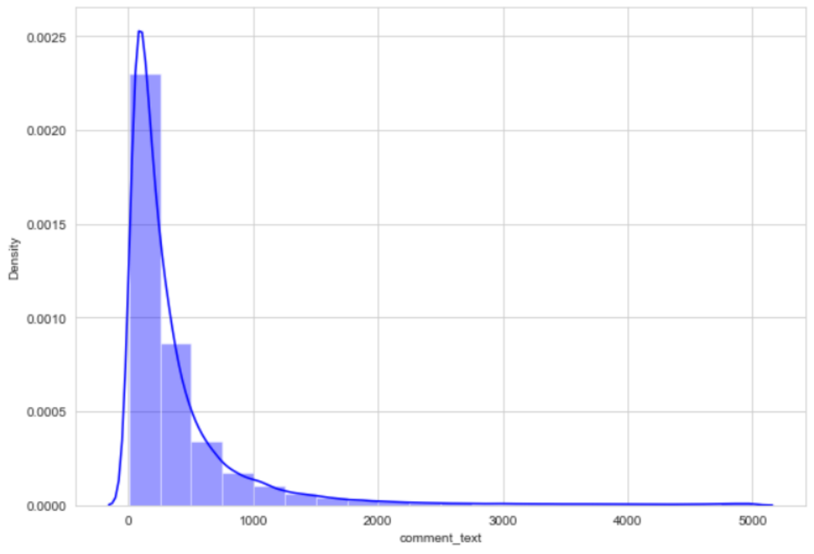


```
distribution.plot.pie(y = 'count', title = 'Label distribution over comments', autopct='%.2f', figsize = (20, 10))\
                             .legend(loc='center left', bbox_to_anchor=(1.3, 0.5));
```



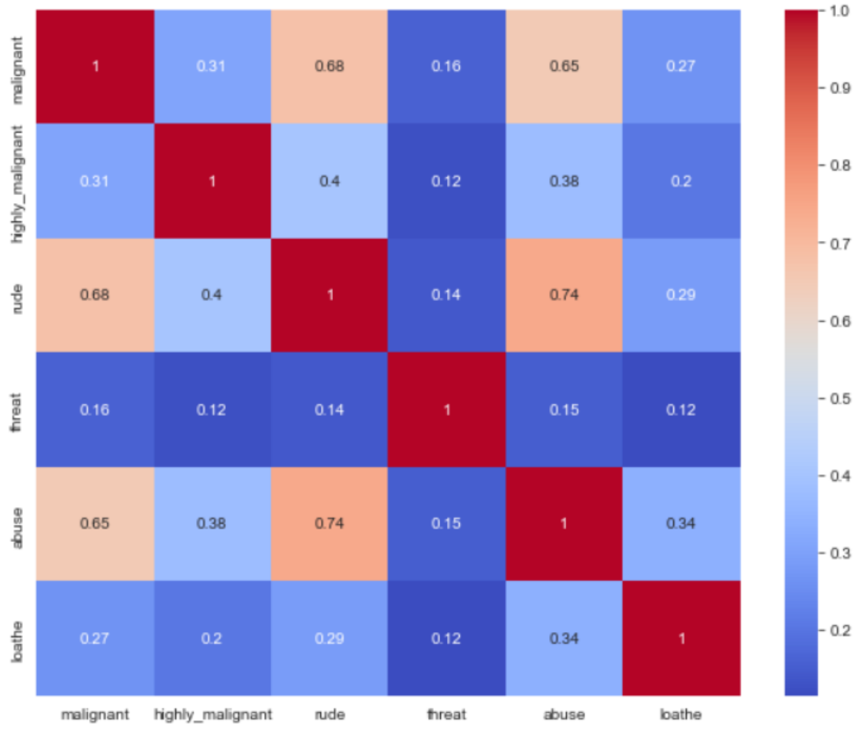
As we see above malignant, and rude sentence are high classified and threat, loathe are least classified.

```
sns.set_style('whitegrid')
plt.figure(figsize=(10,7))
comment_len = df.comment_text.str.len()
sns.distplot(comment_len, bins=20, color = 'blue');
```



We can see that few sentences are really long but most of the sentence are small.

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot = True, cmap= "coolwarm")
plt.show()
```



	malignant	highly_malignant	rude	threat	abuse	loathe
malignant	1	0.31	0.68	0.16	0.65	0.27
highly_malignant	0.31	1	0.4	0.12	0.38	0.2
rude	0.68	0.4	1	0.14	0.74	0.29
threat	0.16	0.12	0.14	1	0.15	0.12
abuse	0.65	0.38	0.74	0.15	1	0.34
loathe	0.27	0.2	0.29	0.12	0.34	1

1. We can see more corelations in the variables, Abuse have more corelation with malignant and rude.
2. Rude has more positive corelation with malignant
3. we don’t have any negative corelations in the data.

Data Sources and their formats

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

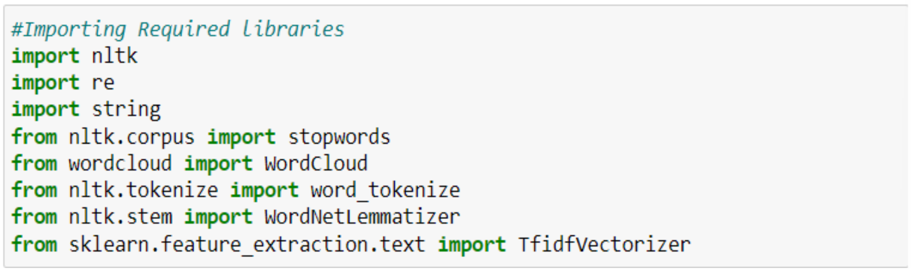
The data set includes:

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

This project is more about exploration, feature engineering and classification that can be done on this data. Since the data set is huge and includes many categories of comments, we can do good amount of data exploration and derive some interesting features using the comments text column available.

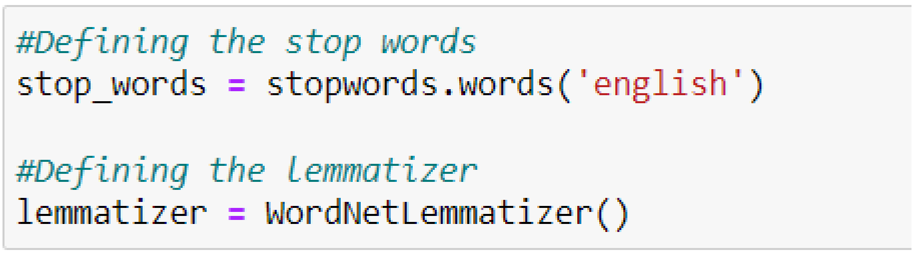
Data Pre-processing Done

I imported all the required libraries for cleansing the data.



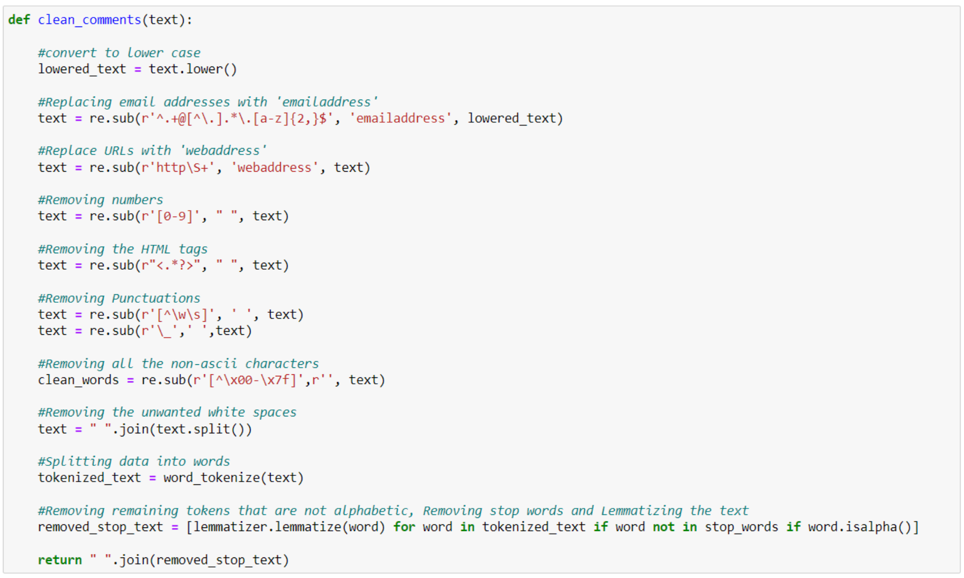
```
#Importing Required libraries
import nltk
import re
import string
from nltk.corpus import stopwords
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

After importing all the required libraries, I have defined stopwords and lemmatize to a variable.



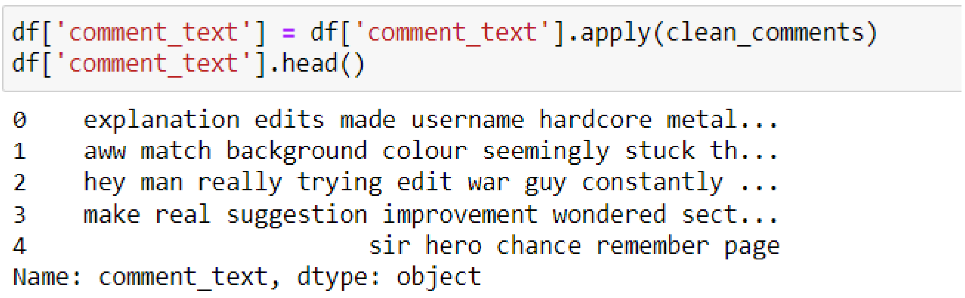
```
#Defining the stop words
stop_words = stopwords.words('english')

#Defining the lemmatizer
lemmatizer = WordNetLemmatizer()
```

Post which I have defined a function for cleaning the data.



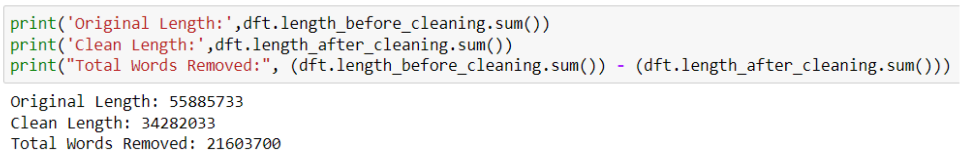
```
def clean_comments(text):

    #convert to lower case
    lowered_text = text.lower()

    #Replacing email addresses with 'emailaddress'
    text = re.sub(r'^.+@[^\.].*\.[a-z]{2,}$', 'emailaddress', lowered_text)

    #Replace URLs with 'webaddress'
    text = re.sub(r'http\S+', 'webaddress', text)

    #Removing numbers
    text = re.sub(r'[0-9]', " ", text)

    #Removing the HTML tags
    text = re.sub(r"<.*?>", " ", text)

    #Removing Punctuations
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\_',' ',text)

    #Removing all the non-ascii characters
    clean_words = re.sub(r'[^\x00-\x7f]',r'', text)

    #Removing the unwanted white spaces
    text = " ".join(text.split())

    #Splitting data into words
    tokenized_text = word_tokenize(text)

    #Removing remaining tokens that are not alphabetic, Removing stop words and Lemmatizing the text
    removed_stop_text = [lemmatizer.lemmatize(word) for word in tokenized_text if word not in stop_words if word.isalpha()]

    return " ".join(removed_stop_text)
```

Post on creating a function I have passed my data into the same to clean it.



```
df['comment_text'] = df['comment_text'].apply(clean_comments)
df['comment_text'].head()
```

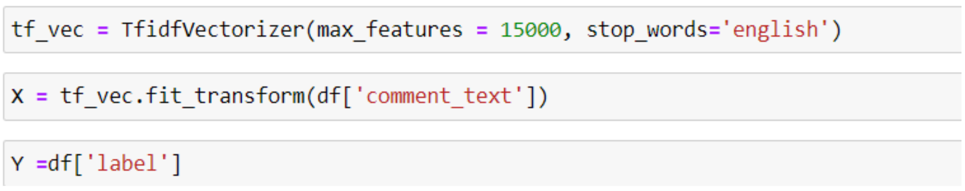
```
0    explanation edits made username hardcore metal...
1    aww match background colour seemingly stuck th...
2    hey man really trying edit war guy constantly ...
3    make real suggestion improvement wondered sect...
4                        sir hero chance remember page
Name: comment_text, dtype: object
```



```
print('Original Length:',dft.length_before_cleaning.sum())
print('Clean Length:',dft.length_after_cleaning.sum())
print("Total Words Removed:", (dft.length_before_cleaning.sum()) - (dft.length_after_cleaning.sum()))
```

```
Original Length: 55885733
Clean Length: 34282033
Total Words Removed: 21603700
```

The total amount of data that is cleansed from the original data is 24418290. Now the data is cleansed and ready for training but before which I converted the data into vectors for the machine learning models to understand the data, so I imported TFIDF vectorizer and I have made the max feature as 15000.



```
tf_vec = TfidfVectorizer(max_features = 15000, stop_words='english')
```

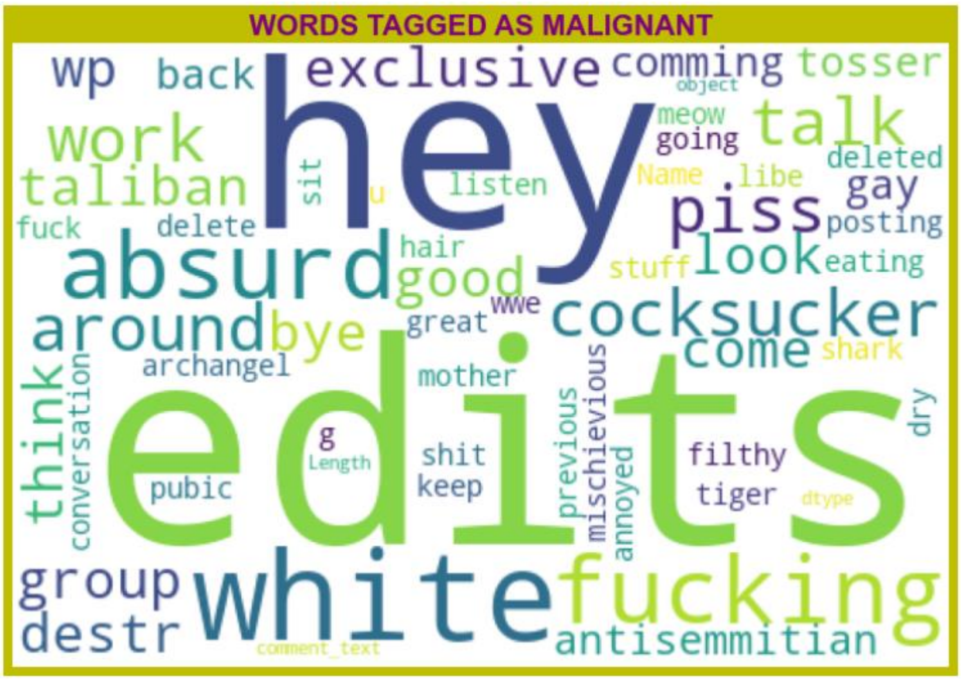
```
X = tf_vec.fit_transform(df['comment_text'])
```

```
Y =df['label']
```

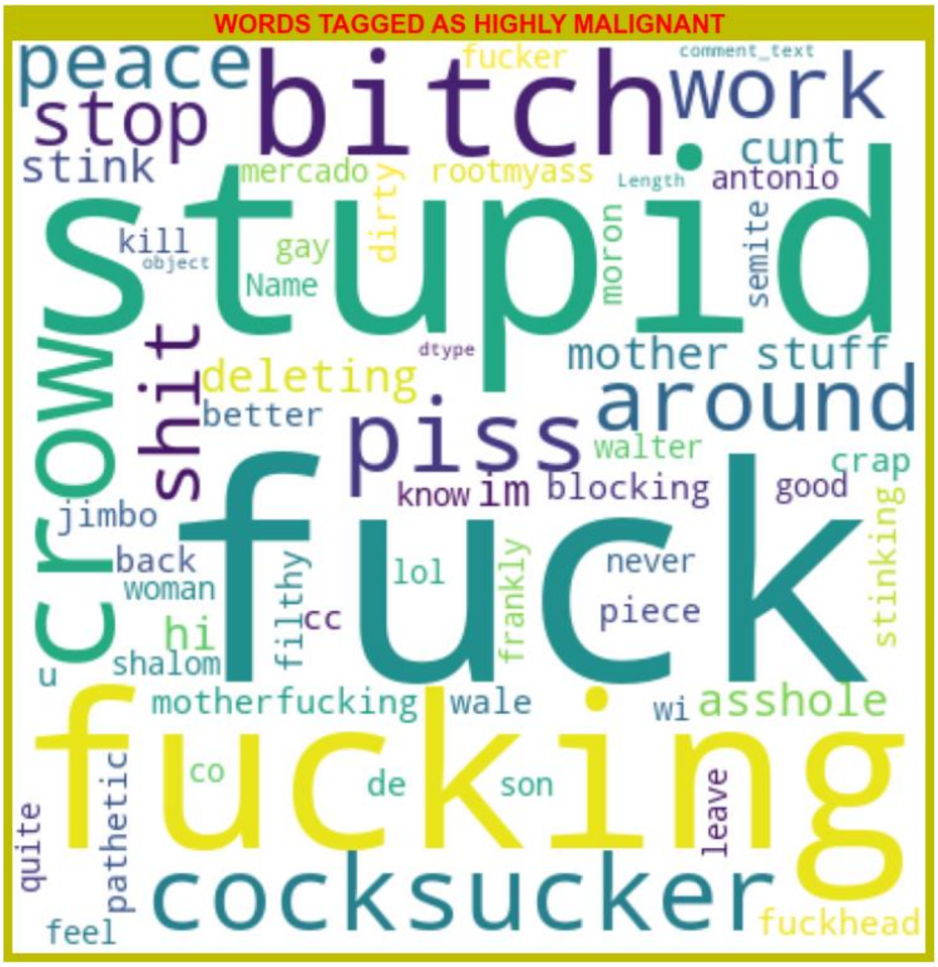
And I have split the data into two parts X and y and made them ready for training. I have created a new feature name label and summed all the other numerical column and changed the output as binary i.e., if the sentence is categorised as malignant it will be 1 or else it will be 0.

Data Inputs- Logic- Output Relationships

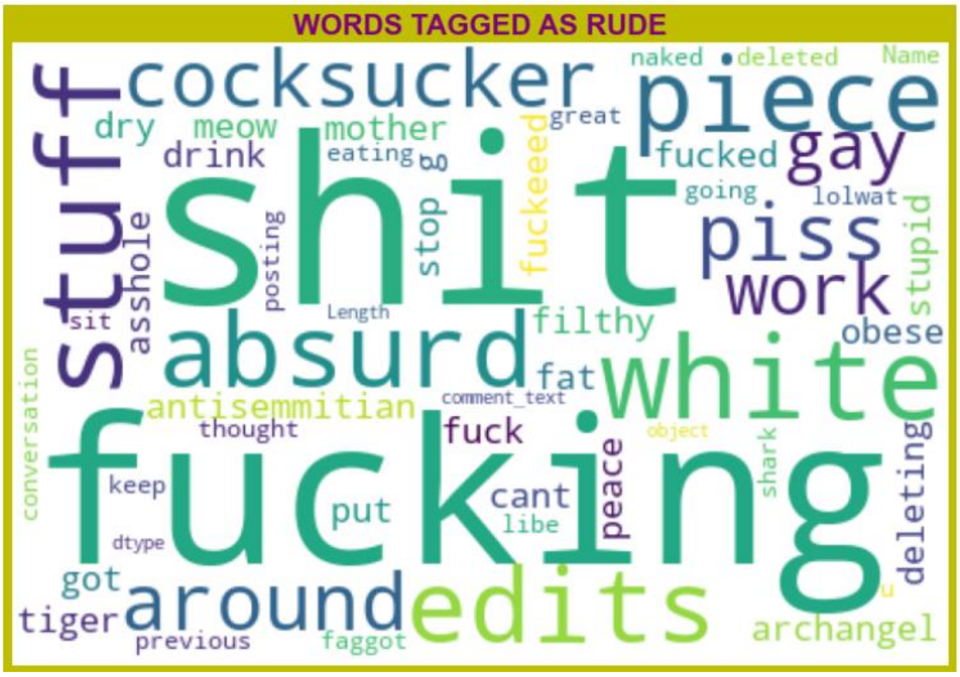
I have analysed the input output logic with word cloud and I have word clouded the sentenced that as classified as foul language in every category.



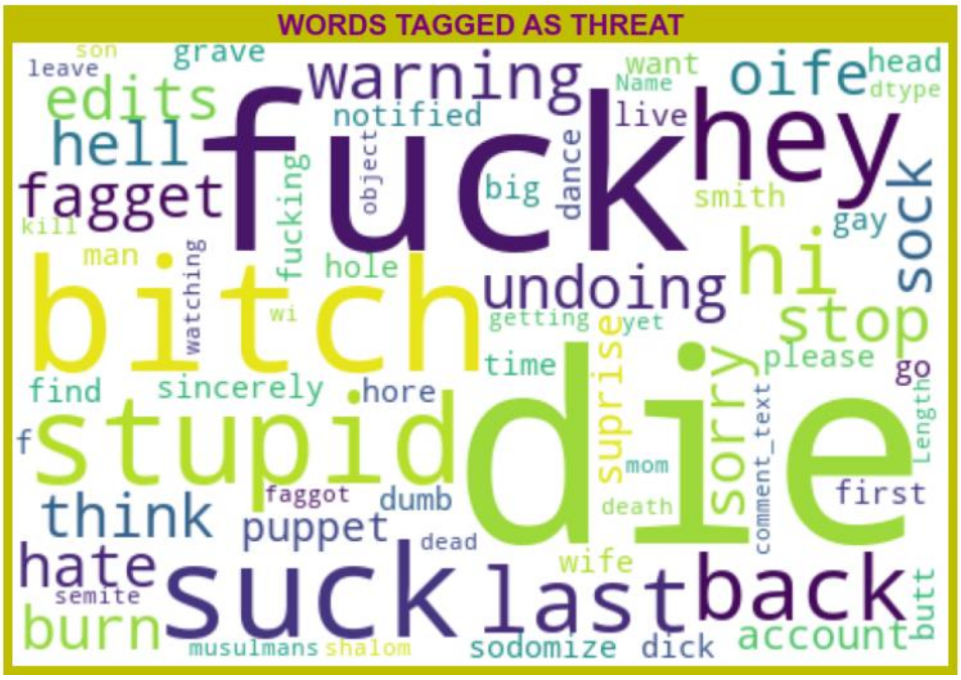
We can see the foul words that are mostly used in malignant classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.



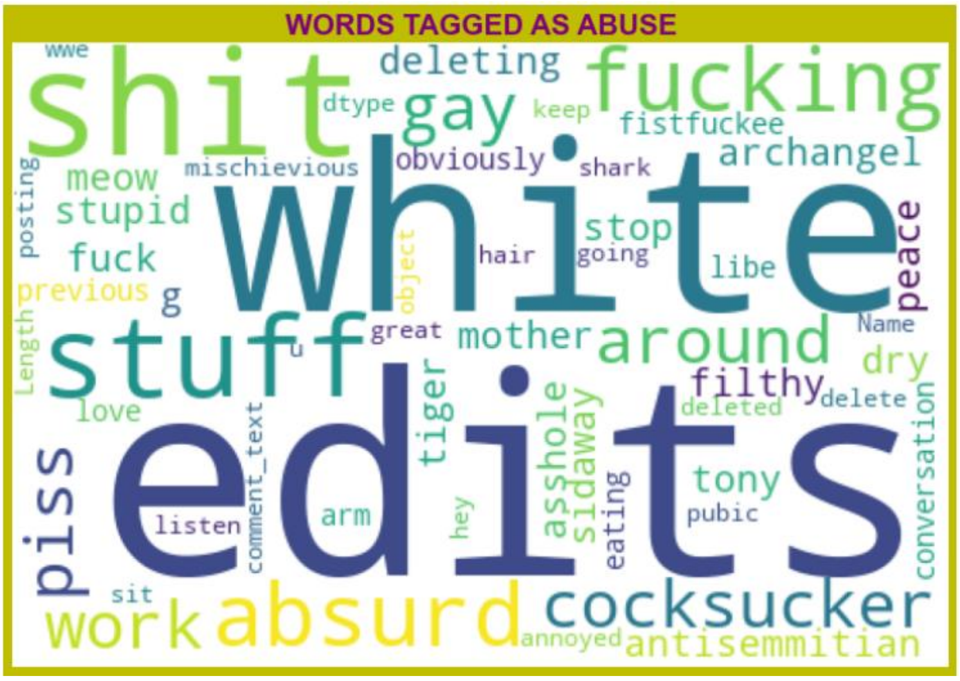
We can see the foul words that are mostly used in highly_malignant classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.



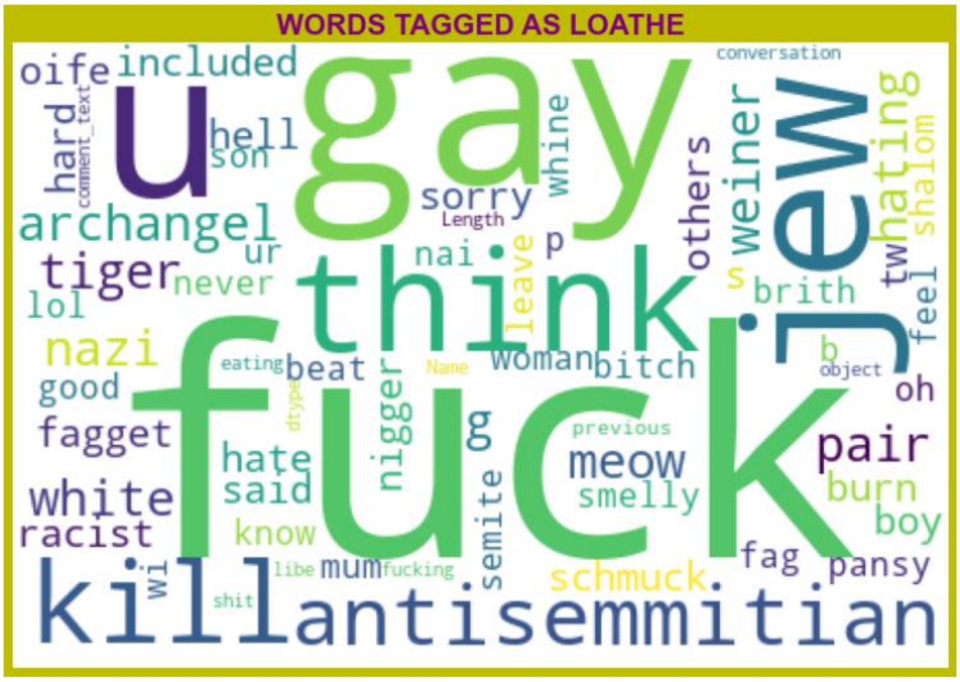
We can see the foul words that are mostly used in rude classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.



We can see the foul words that are mostly used in threat classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.



We can see the foul words that are mostly used in abuse classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.



We can see the foul words that are mostly used in loathe classified sentences we are seeing top 400 words the words which are bigger in size are mostly used.

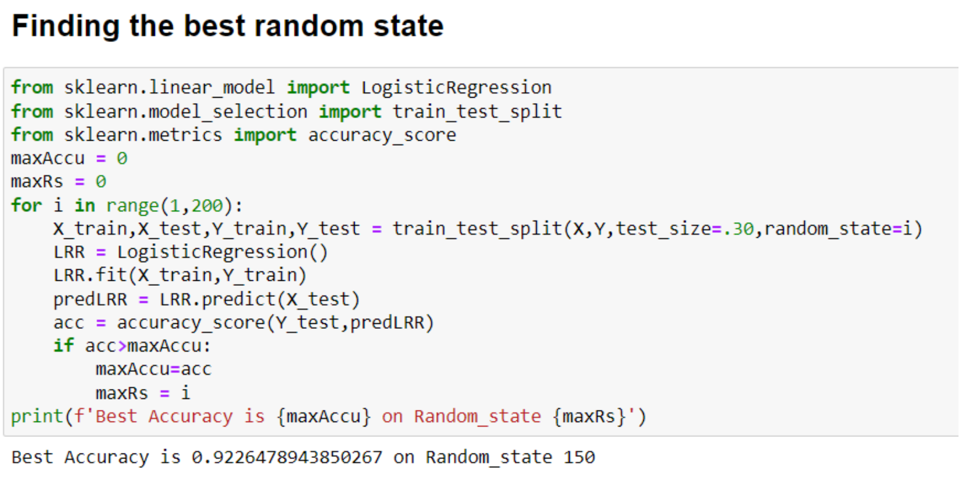
Hardware and Software Requirements and Tools Used

1. Python 3.8.
2. NumPy.
3. Pandas.
4. Matplotlib.
5. Seaborn. 6. Data science.
6. SciPy
7. Sklearn.
8. Anaconda Environment, Jupyter Notebook.

Model/s Development and Evaluation

Testing of Identified Approaches (Algorithms)

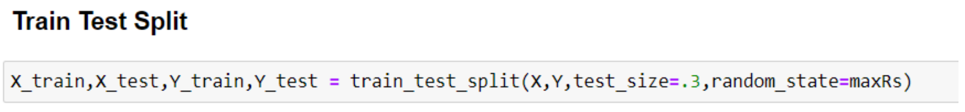
I have started the training in selecting the best random state parameter for the model as follows.



Finding the best random state

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
maxAccu = 0
maxRs = 0
for i in range(1,200):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=.30,random_state=i)
    LRR = LogisticRegression()
    LRR.fit(X_train,Y_train)
    predLRR = LRR.predict(X_test)
    acc = accuracy_score(Y_test,predLRR)
    if acc>maxAccu:
        maxAccu=acc
        maxRs = i
print(f'Best Accuracy is {maxAccu} on Random_state {maxRs}')
```

```
Best Accuracy is 0.9226478943850267 on Random_state 150
```



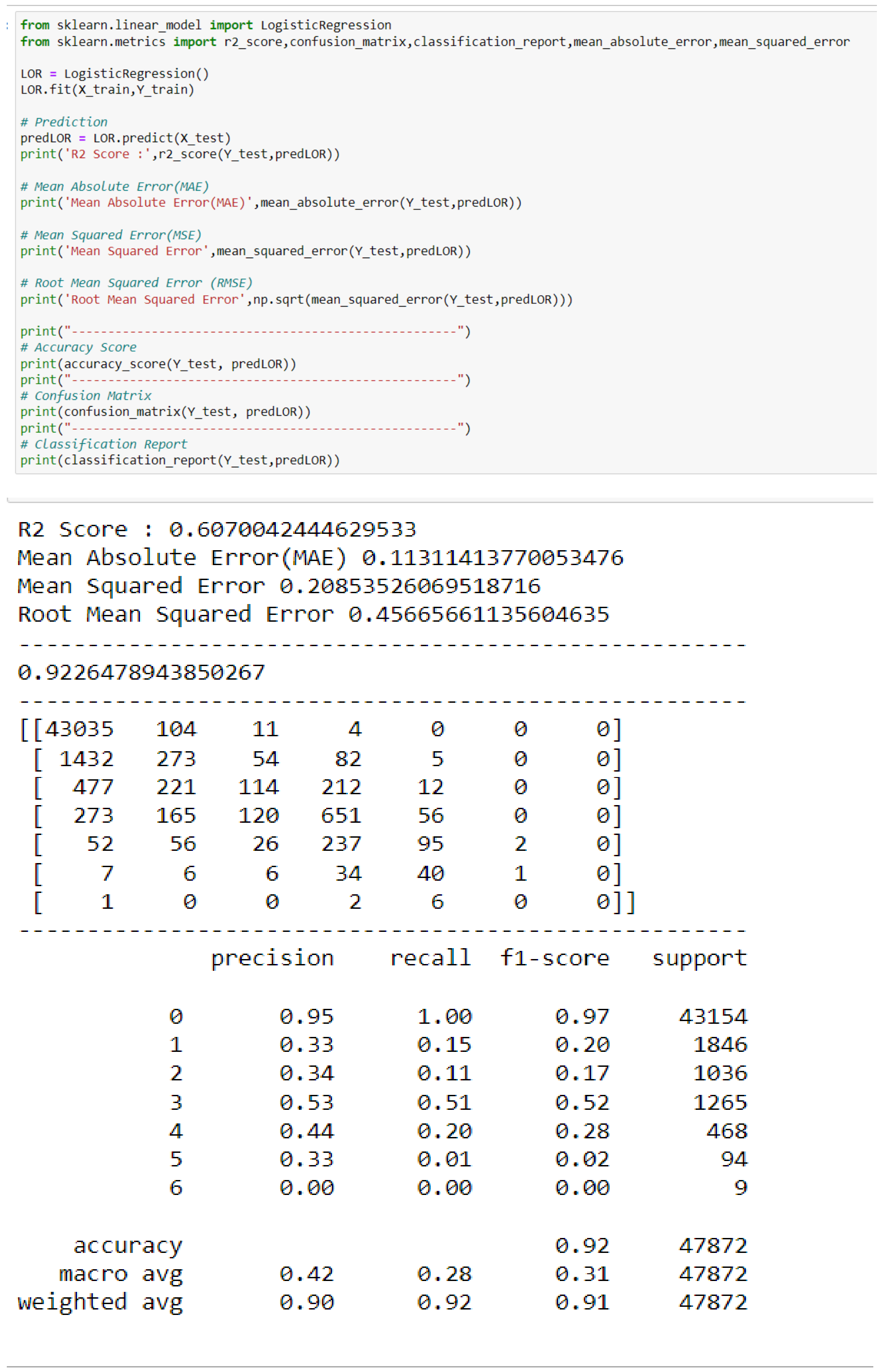
Train Test Split

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=.3,random_state=maxRs)
```

After selecting the best random state parameter, I have spitted the data into test and train with test size as 30 %. Again, I have imported the required libraries to import my ML algorithms.

Run and evaluate selected models

Logistic Regression



```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score,confusion_matrix,classification_report,mean_absolute_error,mean_squared_error

LOR = LogisticRegression()
LOR.fit(X_train,Y_train)

# Prediction
predLOR = LOR.predict(X_test)
print('R2 Score :',r2_score(Y_test,predLOR))

# Mean Absolute Error(MAE)
print('Mean Absolute Error(MAE)',mean_absolute_error(Y_test,predLOR))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predLOR))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predLOR)))

print("----------------------------------------------------")
# Accuracy Score
print(accuracy_score(Y_test, predLOR))
print("----------------------------------------------------")
# Confusion Matrix
print(confusion_matrix(Y_test, predLOR))
print("----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predLOR))
```

```
R2 Score : 0.6070042444629533
Mean Absolute Error(MAE) 0.11311413770053476
Mean Squared Error 0.20853526069518716
Root Mean Squared Error 0.45665661135604635
----------------------------------------------------
0.9226478943850267
----------------------------------------------------
[[43035   104    11     4     0     0     0]
 [ 1432   273    54    82     5     0     0]
 [  477   221   114   212    12     0     0]
 [  273   165   120   651    56     0     0]
 [   52    56    26   237    95     2     0]
 [    7     6     6    34    40     1     0]
 [    1     0     0     2     6     0     0]]
----------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     43154
           1       0.33      0.15      0.20      1846
           2       0.34      0.11      0.17      1036
           3       0.53      0.51      0.52      1265
           4       0.44      0.20      0.28       468
           5       0.33      0.01      0.02        94
           6       0.00      0.00      0.00         9

    accuracy                           0.92     47872
   macro avg       0.42      0.28      0.31     47872
weighted avg       0.90      0.92      0.91     47872
```

Decision Tree Classifier

```python
from sklearn.tree import DecisionTreeClassifier

# Checking accuracy for Decision Tree Classifier
DTC = DecisionTreeClassifier()
DTC.fit(X_train,Y_train)

# [Prediction]
predDTC = DTC.predict(X_test)
print('R2 Score:',r2_score(Y_test,predDTC))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predDTC))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predDTC))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predDTC)))
print("-------------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predDTC))
print("-------------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predDTC))
print("-------------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predDTC))
```

```
R2 Score: 0.5077613014890081
Mean Absolute Error 0.14275568181818182
Mean Squared Error 0.2611965240641711
Root Mean Squared Error 0.5110738929589058
-----------------------------------------------------
Accuracy Score:  0.9011948529411765
-----------------------------------------------------
Confusion Matrix:
 [[41900   913   192   121    24     4     0]
 [ 1068   366   193   184    27     7     1]
 [  317   183   195   271    59    10     1]
 [  194   156   186   559   148    20     2]
 [   35    37    52   218   106    19     1]
 [    8    10     5    26    29    16     0]
 [    1     2     0     4     1     1     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     43154
           1       0.22      0.20      0.21      1846
           2       0.24      0.19      0.21      1036
           3       0.40      0.44      0.42      1265
           4       0.27      0.23      0.25       468
           5       0.21      0.17      0.19        94
           6       0.00      0.00      0.00         9

    accuracy                           0.90     47872
   macro avg       0.33      0.31      0.32     47872
weighted avg       0.90      0.90      0.90     47872
```

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Checking accuracy for Random Forest Classifier
RFC = RandomForestClassifier()
RFC.fit(X_train,Y_train)

# [Prediction]
predRFC = RFC.predict(X_test)
print('R2 Score:',r2_score(Y_test,predRFC))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predRFC))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predRFC))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predRFC)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predRFC))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predRFC))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predRFC))
```

```
R2 Score: 0.5785028994755926
Mean Absolute Error 0.1186288435828877
Mean Squared Error 0.22365892379679145
Root Mean Squared Error 0.4729259178738161
-----------------------------------------------------
Accuracy Score:  0.9200576537433155
-----------------------------------------------------
Confusion Matrix:
 [[42917   138    47    47     5     0     0]
 [ 1399   173   108   155    11     0     0]
 [  472   110   144   293    17     0     0]
 [  278    73   106   736    65     7     0]
 [   66    23    28   277    71     3     0]
 [    9     5     5    42    29     4     0]
 [    1     0     0     6     2     0     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     43154
           1       0.33      0.09      0.15      1846
           2       0.33      0.14      0.20      1036
           3       0.47      0.58      0.52      1265
           4       0.35      0.15      0.21       468
           5       0.29      0.04      0.07        94
           6       0.00      0.00      0.00         9

    accuracy                           0.92     47872
   macro avg       0.39      0.29      0.30     47872
weighted avg       0.89      0.92      0.90     47872
```

KNeighborsClassifier

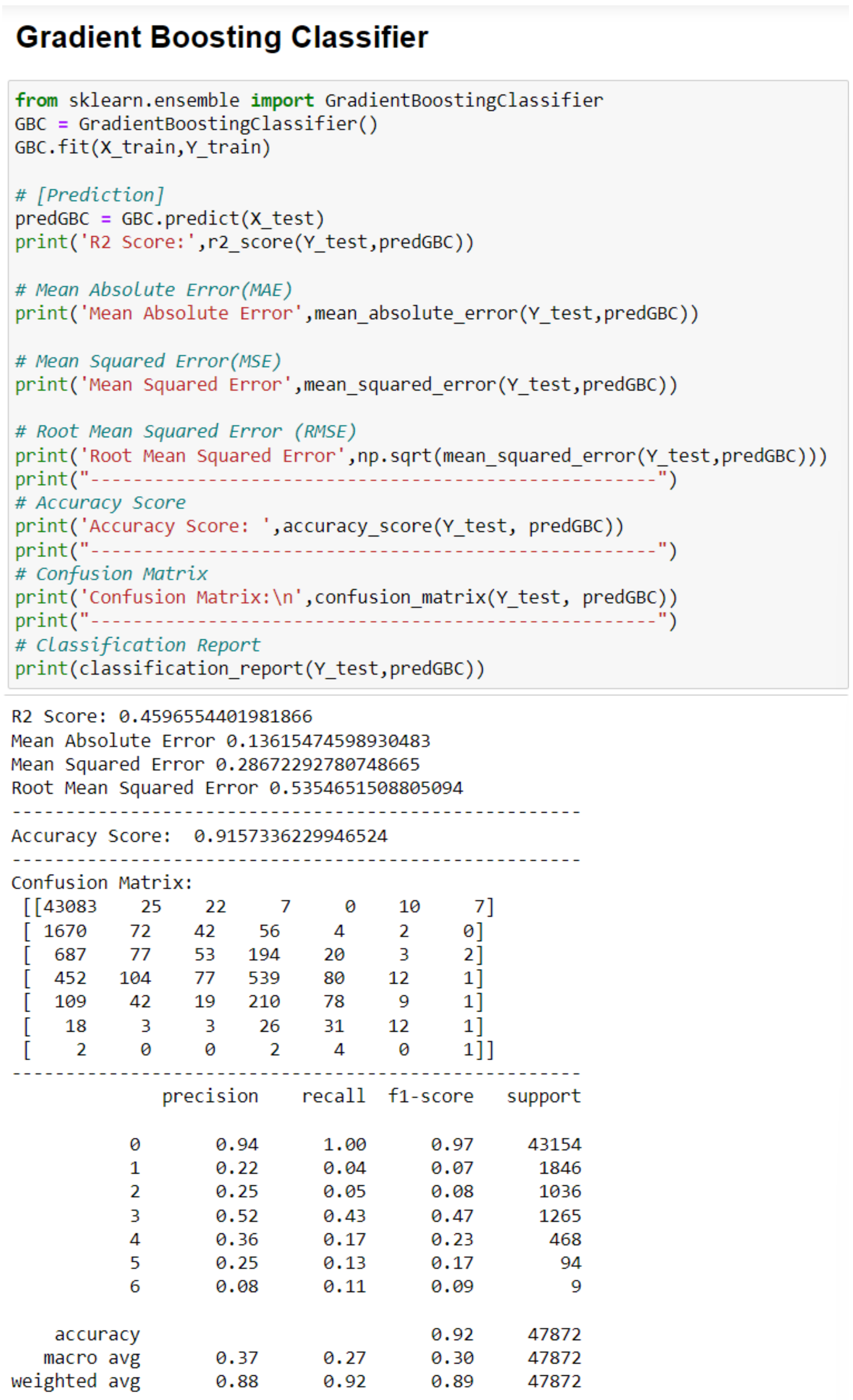
```python
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier()
KNN.fit(X_train,Y_train)

# [Prediction]
predKNN = KNN.predict(X_test)
print('R2 Score:',r2_score(Y_test,predKNN))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predKNN))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predKNN))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predKNN)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predKNN))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predKNN))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predKNN))
```

```
R2 Score: 0.18011393363016415
Mean Absolute Error 0.17599014037433156
Mean Squared Error 0.43505598262032086
Root Mean Squared Error 0.6595877368632022
-----------------------------------------------------
Accuracy Score:  0.9075451203208557
-----------------------------------------------------
Confusion Matrix:
 [[43043    54    38    18     0     1     0]
 [ 1702    75    37    32     0     0     0]
 [  829    41    74    81    11     0     0]
 [  901    30    78   203    53     0     0]
 [  292    14    37    73    50     2     0]
 [   55     0     6    14    18     1     0]
 [    5     0     0     1     3     0     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     43154
           1       0.35      0.04      0.07      1846
           2       0.27      0.07      0.11      1036
           3       0.48      0.16      0.24      1265
           4       0.37      0.11      0.17       468
           5       0.25      0.01      0.02        94
           6       0.00      0.00      0.00         9

    accuracy                           0.91     47872
   macro avg       0.38      0.20      0.22     47872
weighted avg       0.86      0.91      0.88     47872
```

AdaBoost Classifier

```python
from sklearn.ensemble import AdaBoostClassifier
ADA = AdaBoostClassifier()
ADA.fit(X_train,Y_train)

# [Prediction]
predADA = ADA.predict(X_test)
print('R2 Score:',r2_score(Y_test,predADA))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predADA))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predADA))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predADA)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predADA))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predADA))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predADA))
```

```
R2 Score: 0.3402174834417607
Mean Absolute Error 0.15478776737967914
Mean Squared Error 0.35010026737967914
Root Mean Squared Error 0.5916927136442354
-----------------------------------------------------
Accuracy Score:  0.9115140374331551
-----------------------------------------------------
Confusion Matrix:
 [[43088     7     4    52     1     2     0]
 [ 1740    12     0    90     1     3     0]
 [  789     9     2   227     7     2     0]
 [  729     7     3   477    38    11     0]
 [  165     0     1   234    52    15     1]
 [   30     1     1    42    15     5     0]
 [    1     0     0     4     1     3     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     43154
           1       0.33      0.01      0.01      1846
           2       0.18      0.00      0.00      1036
           3       0.42      0.38      0.40      1265
           4       0.45      0.11      0.18       468
           5       0.12      0.05      0.07        94
           6       0.00      0.00      0.00         9

    accuracy                           0.91     47872
   macro avg       0.35      0.22      0.23     47872
weighted avg       0.87      0.91      0.88     47872
```



Gradient Boosting Classifier

```python
from sklearn.ensemble import GradientBoostingClassifier
GBC = GradientBoostingClassifier()
GBC.fit(X_train,Y_train)

# [Prediction]
predGBC = GBC.predict(X_test)
print('R2 Score:',r2_score(Y_test,predGBC))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predGBC))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predGBC))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predGBC)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predGBC))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predGBC))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predGBC))
```

```
R2 Score: 0.4596554401981866
Mean Absolute Error 0.13615474598930483
Mean Squared Error 0.28672292780748665
Root Mean Squared Error 0.5354651508805094
-----------------------------------------------------
Accuracy Score:  0.9157336229946524
-----------------------------------------------------
Confusion Matrix:
 [[43083    25    22     7     0    10     7]
 [ 1670    72    42    56     4     2     0]
 [  687    77    53   194    20     3     2]
 [  452   104    77   539    80    12     1]
 [  109    42    19   210    78     9     1]
 [   18     3     3    26    31    12     1]
 [    2     0     0     2     4     0     1]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.94      1.00      0.97     43154
           1       0.22      0.04      0.07      1846
           2       0.25      0.05      0.08      1036
           3       0.52      0.43      0.47      1265
           4       0.36      0.17      0.23       468
           5       0.25      0.13      0.17        94
           6       0.08      0.11      0.09         9

    accuracy                           0.92     47872
   macro avg       0.37      0.27      0.30     47872
weighted avg       0.88      0.92      0.89     47872
```

XGBClassifier

```
from xgboost import XGBClassifier
XGB = XGBClassifier()
XGB.fit(X_train,Y_train)

# [Prediction]
predXGB = XGB.predict(X_test)
print('R2 Score:',r2_score(Y_test,predXGB))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predXGB))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predXGB))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predXGB)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predXGB))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predXGB))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predXGB))
```

```
R2 Score: 0.5773612710161542
Mean Absolute Error 0.11877506684491979
Mean Squared Error 0.22426470588235295
Root Mean Squared Error 0.47356594670895935
-----------------------------------------------------
Accuracy Score:  0.9202665441176471
-----------------------------------------------------
Confusion Matrix:
 [[43045    79    16    13     0     1     0]
 [ 1510   176    60    88    12     0     0]
 [  517   150   110   238    20     1     0]
 [  306   145   102   603    97    12     0]
 [   60    42    27   213   112    14     0]
 [    9     8     5    27    36     9     0]
 [    1     0     0     3     4     1     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     43154
           1       0.29      0.10      0.14      1846
           2       0.34      0.11      0.16      1036
           3       0.51      0.48      0.49      1265
           4       0.40      0.24      0.30       468
           5       0.24      0.10      0.14        94
           6       0.00      0.00      0.00         9

    accuracy                           0.92     47872
   macro avg       0.39      0.29      0.32     47872
weighted avg       0.89      0.92      0.90     47872
```

Support Vector Machine classifier

```python
from sklearn.svm import SVC

# Checking accuracy for Support Vector Machine Classifier
svc = SVC()
svc.fit(X_train,Y_train)

# [Prediction]
predsvc = svc.predict(X_test)
print('R2 Score:',r2_score(Y_test,predsvc))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predsvc))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predsvc))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predsvc)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predsvc))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predsvc))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predsvc))
```

```
R2 Score: 0.6024377306251993
Mean Absolute Error 0.11336480614973261
Mean Squared Error 0.21095838903743316
Root Mean Squared Error 0.45930206731238776
-----------------------------------------------------
Accuracy Score:  0.9228567847593583
-----------------------------------------------------
Confusion Matrix:
 [[43068    68    14     4     0     0     0]
 [ 1495   204    49    94     4     0     0]
 [  513   165   104   244    10     0     0]
 [  296   114    95   698    62     0     0]
 [   56    43    19   244   104     2     0]
 [   10     3     6    32    42     1     0]
 [    1     0     0     2     6     0     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     43154
           1       0.34      0.11      0.17      1846
           2       0.36      0.10      0.16      1036
           3       0.53      0.55      0.54      1265
           4       0.46      0.22      0.30       468
           5       0.33      0.01      0.02        94
           6       0.00      0.00      0.00         9

    accuracy                           0.92     47872
   macro avg       0.42      0.28      0.31     47872
weighted avg       0.89      0.92      0.90     47872
```

Extra Trees Classifier

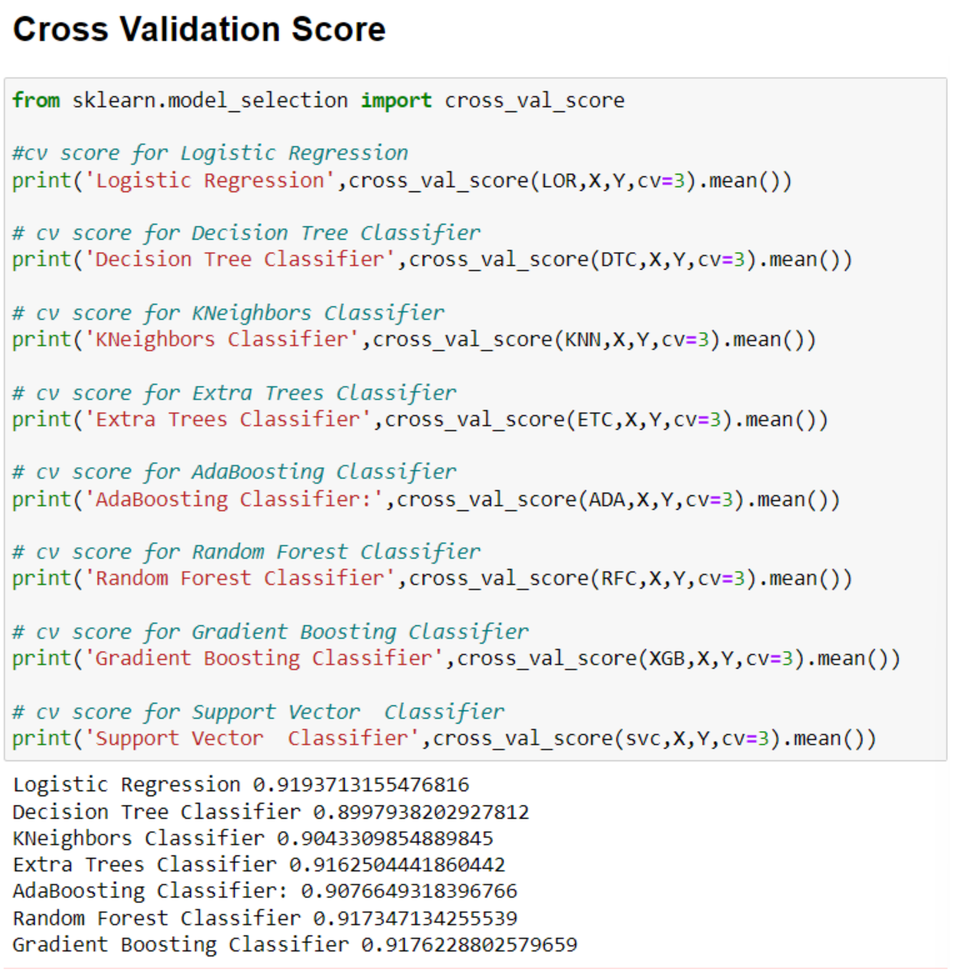
```
from sklearn.ensemble import ExtraTreesClassifier

# Checking accuracy for Support Vector Machine Classifier
ETC = ExtraTreesClassifier()
ETC.fit(X_train,Y_train)

# [Prediction]
predETC = ETC.predict(X_test)
print('R2 Score:',r2_score(Y_test,predETC))

# Mean Absolute Error(MAE)
print('Mean Absolute Error',mean_absolute_error(Y_test,predETC))

# Mean Squared Error(MSE)
print('Mean Squared Error',mean_squared_error(Y_test,predETC))

# Root Mean Squared Error (RMSE)
print('Root Mean Squared Error',np.sqrt(mean_squared_error(Y_test,predETC)))
print("-----------------------------------------------------")
# Accuracy Score
print('Accuracy Score: ',accuracy_score(Y_test, predETC))
print("-----------------------------------------------------")
# Confusion Matrix
print('Confusion Matrix:\n',confusion_matrix(Y_test, predETC))
print("-----------------------------------------------------")
# Classification Report
print(classification_report(Y_test,predETC))
```

```
R2 Score: 0.5590164826679358
Mean Absolute Error 0.12103108288770054
Mean Squared Error 0.23399899732620322
Root Mean Squared Error 0.4837344285103172
-----------------------------------------------------
Accuracy Score:  0.9200785427807486
-----------------------------------------------------
Confusion Matrix:
 [[42942   131    49    31     1     0     0]
 [ 1430   175    93   134    14     0     0]
 [  514    90   141   273    18     0     0]
 [  327    70   102   692    69     5     0]
 [   75    24    26   246    92     5     0]
 [   14     3     5    40    28     4     0]
 [    1     0     0     4     4     0     0]]
-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     43154
           1       0.35      0.09      0.15      1846
           2       0.34      0.14      0.19      1036
           3       0.49      0.55      0.52      1265
           4       0.41      0.20      0.27       468
           5       0.29      0.04      0.07        94
           6       0.00      0.00      0.00         9

    accuracy                           0.92     47872
   macro avg       0.40      0.29      0.31     47872
weighted avg       0.89      0.92      0.90     47872
```

As we can see above, I have imported 8 classification algorithms and I am going to shortlist the best amongst these in basis of accuracy precision recall and F1 scores.

So, like above I have run all the algorithms with the data.

Key-Metrics for success in solving problem under consideration.

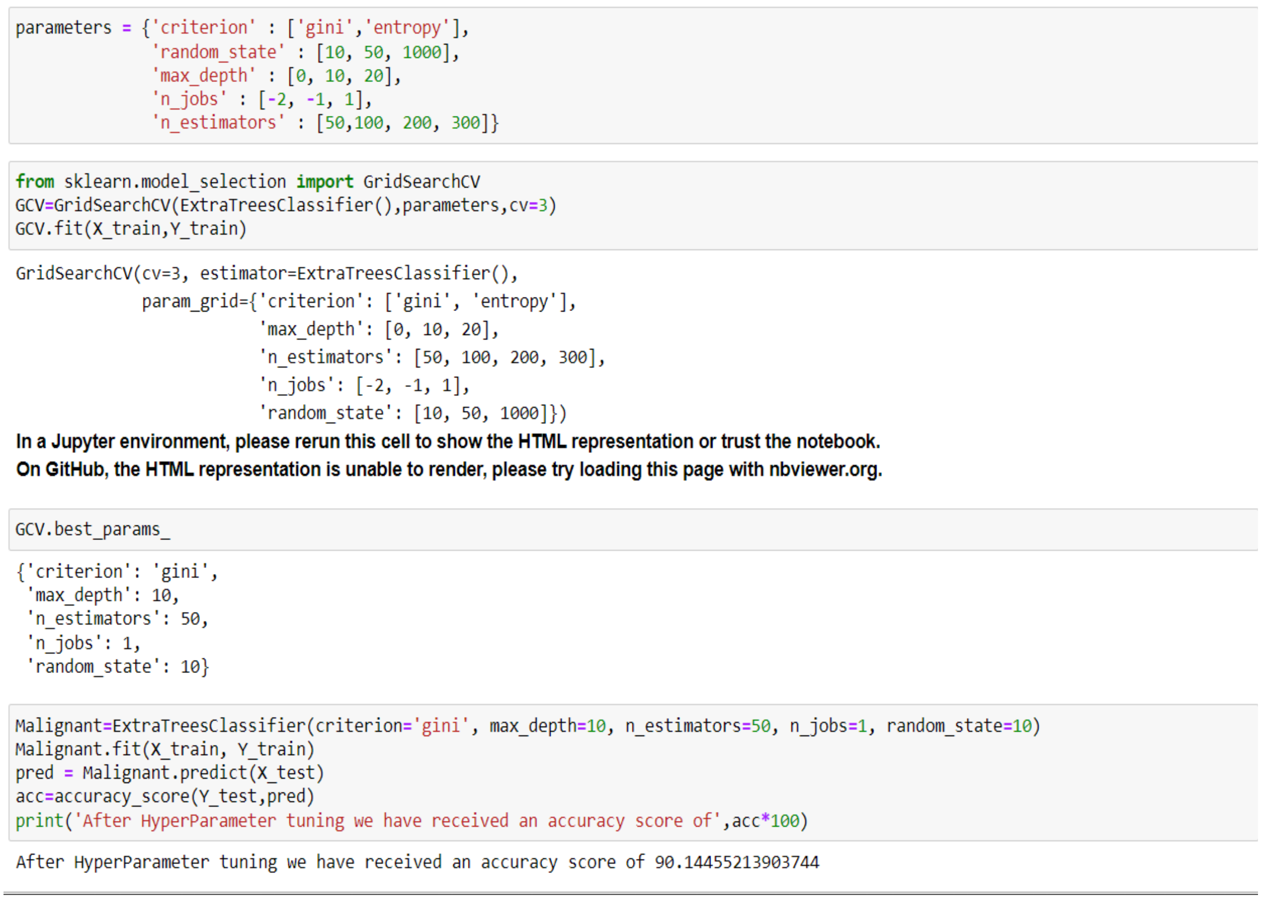


Cross Validation Score

```python
from sklearn.model_selection import cross_val_score

#cv score for Logistic Regression
print('Logistic Regression',cross_val_score(LOR,X,Y,cv=3).mean())

# cv score for Decision Tree Classifier
print('Decision Tree Classifier',cross_val_score(DTC,X,Y,cv=3).mean())

# cv score for KNeighbors Classifier
print('KNeighbors Classifier',cross_val_score(KNN,X,Y,cv=3).mean())

# cv score for Extra Trees Classifier
print('Extra Trees Classifier',cross_val_score(ETC,X,Y,cv=3).mean())

# cv score for AdaBoosting Classifier
print('AdaBoosting Classifier:',cross_val_score(ADA,X,Y,cv=3).mean())

# cv score for Random Forest Classifier
print('Random Forest Classifier',cross_val_score(RFC,X,Y,cv=3).mean())

# cv score for Gradient Boosting Classifier
print('Gradient Boosting Classifier',cross_val_score(XGB,X,Y,cv=3).mean())

# cv score for Support Vector  Classifier
print('Support Vector  Classifier',cross_val_score(svc,X,Y,cv=3).mean())
```

```
Logistic Regression 0.9193713155476816
Decision Tree Classifier 0.8997938202927812
KNeighbors Classifier 0.9043309854889845
Extra Trees Classifier 0.9162504441860442
AdaBoosting Classifier: 0.9076649318396766
Random Forest Classifier 0.917347134255539
Gradient Boosting Classifier 0.9176228802579659
```

Hyperparameter Tuning



```
parameters = {'criterion' : ['gini','entropy'],
              'random_state' : [10, 50, 1000],
              'max_depth' : [0, 10, 20],
              'n_jobs' : [-2, -1, 1],
              'n_estimators' : [50,100, 200, 300]}
```

```
from sklearn.model_selection import GridSearchCV
GCV=GridSearchCV(ExtraTreesClassifier(),parameters,cv=3)
GCV.fit(X_train,Y_train)
```

```
GridSearchCV(cv=3, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [0, 10, 20],
                         'n_estimators': [50, 100, 200, 300],
                         'n_jobs': [-2, -1, 1],
                         'random_state': [10, 50, 1000]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

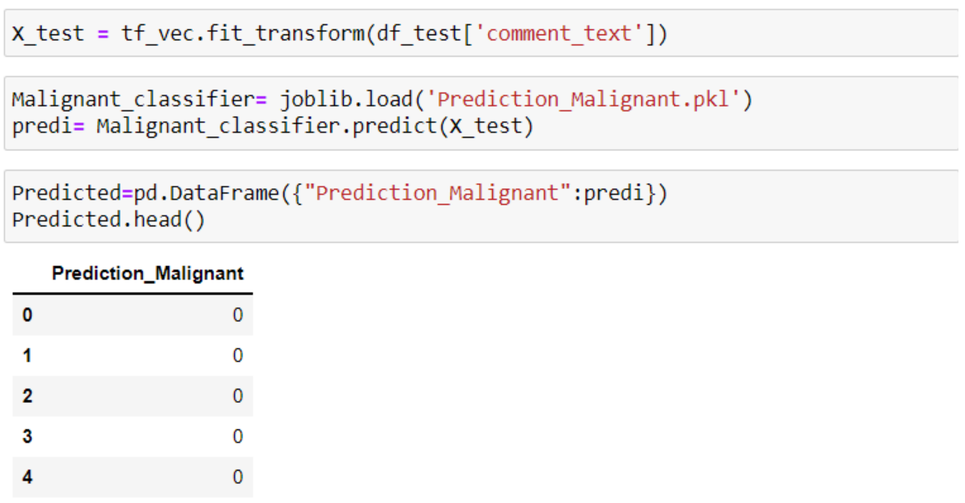
```
GCV.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 10,
 'n_estimators': 50,
 'n_jobs': 1,
 'random_state': 10}
```

```
Malignant=ExtraTreesClassifier(criterion='gini', max_depth=10, n_estimators=50, n_jobs=1, random_state=10)
Malignant.fit(X_train, Y_train)
pred = Malignant.predict(X_test)
acc=accuracy_score(Y_test,pred)
print('After HyperParameter tuning we have received an accuracy score of',acc*100)
```

```
After HyperParameter tuning we have received an accuracy score of 90.14455213903744
```

As we can see above ExtraTrees Classifier tops the chart, I have selected ExtraTrees Classifier model as my final model and I have saved the same for the further usage. Further I have imported the test data and have done prediction on the same.



```
X_test = tf_vec.fit_transform(df_test['comment_text'])
```

```
Malignant_classifier= joblib.load('Prediction_Malignant.pkl')
predi= Malignant_classifier.predict(X_test)
```

```
Predicted=pd.DataFrame({"Prediction_Malignant":predi})
Predicted.head()
```

	Prediction_Malignant
0	0
1	0
2	0
3	0
4	0

CONCLUSION

Key Findings and Conclusions of the Study

The finding of the study is that only few users over online use unparliamentary language. And most of these sentences have more stop words, and are being long. As discussed before few motivated disrespectful crowds uses these foul languages in the online forum to bully the people around and to stop them from doing the things that they are supposed to do. Our Study helps the online forms and social media to induce a ban to profanity or usage of profanity over these forms.

Learning Outcomes of the Study in respect of Data Science

The use of social media is the most common trend among the activities of today's people. Social networking sites offer today's teenagers a platform for communication and entertainment. They use social media to collect more information from their friends and followers. The vastness of social media sites ensures that not all of them provide a decent environment for children. In such cases, the impact of the negative influences of social media on teenage users increases with an increase in the use of **offensive language** in social conversations. This increase could lead to **frustration**, **depression** and a large change in their behaviour. Hence, I propose a novel approach to classify bad language usage in text conversations. I have considered the English medium for textual conversation. I have developed our system based on a foul language classification approach; it is based on an improved version of a Random Forest Classification Algorithm that detects offensive language usage in a conversation. As per our evaluation, we found that lesser number of users conversation is not decent all the time. We trained 159571 observations for eight context categories using a Random Forest algorithm for context detection. Then, the system classifies the use of foul language in one of the trained contexts in the text conversation. In our testbed, we observed 10% of participants used foul language during their text conversation. Hence, our proposed approach can identify the impact of foul language in text conversations using a classification technique and emotion detection to identify the foul language usage

Limitations of this work and Scope for Future Work

The limitation of the study is that we have a imbalanced data so our model learnt more about the non-abusive sentence more than the abusive sentence. Which makes our model act like a overfit model when tested with live data. And also, model tend to not identify a foul or a sarcastically foul language.