FLIP ROBO

MALIGNANT COMMENTS CLASSIFIER PROJECT

Submitted by:

SATYAM GUPTA

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Flip Robo Technologies, Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I want to thank my SME Mr. Khushboo Garg for providing the Dataset and helping us to solve the problem and addressing out our Query in right time.

I would like to express my gratitude towards my parents & members of Flip Robo for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to our institute DataTrained & others seen unseen hands which have given us direct & indirect help in completion of this project. With help of their brilliant guidance and encouragement, I was able to complete my tasks properly and were up to the mark in all the tasks assigned. During the process, I got a chance to see the stronger side of my technical and non-technical aspects and also strengthen my concepts.

INTRODUCTION

Problem Statement

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.

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.

This is a multi-label classification problem.

Multilabel vs Multiclass classification

In multi-class classification, the data can belong to only one label out of all the labels we have. For example, a given picture of an animal may be a cat, dog or elephant only and not a combination of these.

In multi-label classification, data can belong to more than one label simultaneously. For example, in our case a comment may be malignant, threat or loathe at the same time. It may also happen that the comment is positive/neutral and hence does not belong to any of the six labels.

Conceptual Background of the Domain Problem

In the past few years, it's seen that the cases related to social media hatred have increased exponentially. Social media is turning into a dark venomous pit for people nowadays. Online hate is the result of difference in opinion, race, religion, occupation, nationality etc. In social media the people spreading or involved in such kind of activities use filthy languages, aggression, images etc. to offend and gravely hurt the person on the other side. This is one of the major concerns now.

Review of Literature

Sentiment classification regarding toxicity has been intensively researched in the past few years, largely in the context of social media data where researchers have applied various machine learning systems to try and tackle the problem of toxicity as well as the related, more well-known task of sentiment analysis. Comment abuse classification research begins with combining of TF-IDF with sentiment/contextual features. The motivation for our project is to build a model that can detect toxic comments and find the bias with respect to the mention of select identities.

Motivation for the Problem Undertaken

The upsurge in the volume of unwanted comments called malignant comments has created an intense need for the development of more dependable and robust malignant comments filters. Machine learning methods of recent are being used to successfully detect and filter malignant comments. Build a model which can be used to predict in terms of a probability for comments to be malignant. In this case, Label ‘1’ indicates that the comment is malignant, while, Label ‘0’ indicates that the comment is not malignant.

ANALYTICAL PROBLEM FRAMING

Model Building Phase

You need to build a machine learning model. Before model building do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps like-

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

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:

- **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 hatefuland 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.

Both train and test csv(s) are loaded respectively, where, in training dataset, the **independent variable is Comment text** which is of ‘object’type and rest **6 categories or labels are the dependent features** whose values needs to be predicted, are of Boolean in nature being ‘int64’ type.

The sample datasets are given below:

Train dataset

	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

Test dataset

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

Data Pre-processing

- **Checking the value counts of the features**

```
df_train['malignant'].value_counts()
```

```
0    144277
1     15294
Name: malignant, dtype: int64
```

```
df_train['highly_malignant'].value_counts()
```

```
0    157976
1      1595
Name: highly_malignant, dtype: int64
```

```
df_train['rude'].value_counts()
```

```
0    151122
1      8449
Name: rude, dtype: int64
```

```
df_train['threat'].value_counts()
```

```
0    159093
1       478
Name: threat, dtype: int64
```

```
df_train['abuse'].value_counts()
```

```
0    151694
1      7877
Name: abuse, dtype: int64
```

```
df_train['loathe'].value_counts()
```

```
0    158166
1      1405
```

- **Checking for null values**

```
df_train.isnull().sum()    #Checking for null values in the train dataset
```

```
id                 0
comment_text       0
malignant          0
highly_malignant   0
rude               0
threat             0
abuse              0
loathe             0
dtype: int64
```

```
#Checking the datatype of the testing dataset
df_test.dtypes
```

```
id              object
comment_text    object
dtype: object
```

```
df_test.isnull().sum()    #Checking for null values in the test dataset
```

```
id              0
comment_text    0
dtype: int64
```

We can see that there are no null values present in both training and testing datasets.

- **Checking statistical summary of the dataset**

```
df_train.describe()    #Statistical summary of the dataset
```

	malignant	highly_malignant	rude	threat	abuse	loathe
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

- The minimum value and the maximum value of the attributes is same i.e., 0 and 1 respectively.
- The mean and standard deviation is nearly 0-1 of all the attributes in the training dataset.
- Here, with this statistical analysis, it is interpreted that there are no outliers as well as skewness present in this training dataset.

- The count of each field is equal which shows that there are no missing values present.

- **Finding relationship among data using correlation**

```python
#Checking correlation of the dataset
corr=df_train.corr()  #corr() function provides the correlation value of each column
corr
```

	malignant	highly_malignant	rude	threat	abuse	loathe
malignant	1.000000	0.308619	0.676515	0.157058	0.647518	0.266009
highly_malignant	0.308619	1.000000	0.403014	0.123601	0.375807	0.201600
rude	0.676515	0.403014	1.000000	0.141179	0.741272	0.286867
threat	0.157058	0.123601	0.141179	1.000000	0.150022	0.115128
abuse	0.647518	0.375807	0.741272	0.150022	1.000000	0.337736
loathe	0.266009	0.201600	0.286867	0.115128	0.337736	1.000000

```python
#Plotting heatmap for visualizing the correlation
plt.figure(figsize=(10,8))
sns.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',cmap='YlGn_r',annot=True)
plt.show()
```

- The highest positive correlation is seen in between fields 'rude' and 'abuse'.
- Attribute 'threat' is negatively correlated with each and every other feature of this training dataset.
- Overall the correlation among the attributes is not positive.

- **Dropping the Column**

Due to the wide range of given data, it is extremely fruitful to clean, shape and set the data in the most suitable form. Dropping unnecessary columns declines the chances of producing errors. Thus, column ‘ID’ was dropped from the train dataset as every comment has its own unique id. After dropping the same, the dataset is now having 7 attributes in total including the target variables.

- **Cleaning the data using NLP**

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. This data is usually not necessary or helpful when it comes to analysing data because it may hinder the process or provide inaccurate results.

Before cleaning the data, a new column is created named ‘length_before_cleaning’ which shows the total length of the comments respectively before cleaning the text.

The following steps were taken in order to clean the text:

- Replaced the extra lines or ‘\n’ from the text.
- Transform the text into lower case.
- Replaced the email addresses with the text 'emailaddress'
- Replaced the URLs with the text 'webaddress'
- Removed the numbers
- Removed the HTML tags
- Removed the punctuations
- Removed all the non-ascii characters
- Removed the unwanted white spaces
- Removed the remaining tokens that are not alphabetic
- Removed the stop words

The code for the above-mentioned steps is mentioned below:

```
df_train['comment_text'] = df_train['comment_text'].replace('\n',' ')
```

```
#Function Definition for using regex operations and other text preprocessing for getting cleaned texts
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)
```

All the text cleaning or the above steps are performed by defining a function and applying the same using apply() to the comment_text column of the train dataset.

Tokenization

Word tokenization is the process of splitting a large sample of text into words. This is a requirement in natural language processing tasks where each word needs to be captured and subjected to further analysis.

After cleaning the text, each comment i.e., the corpus is split into words. Thus, the text is tokenized into words using word_tokenize().

Lemmatization

Lemmatization in NLTK refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word known as the lemma.

The NLTK Lemmatization method is based on WorldNet's built-in morph function. Thus, the words are lemmatized using WordNetLemmatizer() after importing the necessary library to perform the same and then creating the instance for it.

- After cleaning the text, a new column is created named 'len_after_cleaning' representing the length of each comment respectively in a column 'comment_text' after doing the required cleaning of the text. With this, it’s come to know about how much data is cleaned.

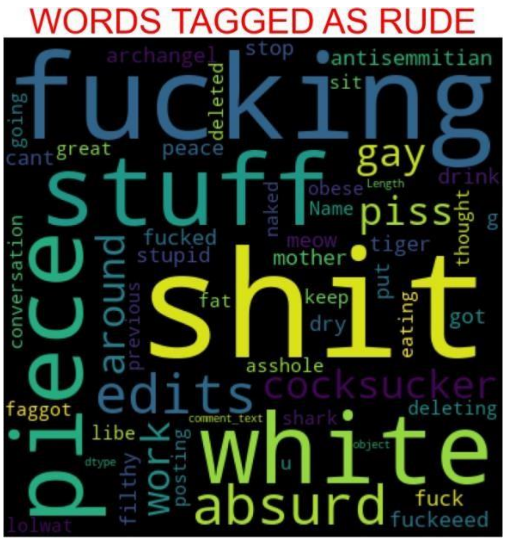
```
Origional Length: 62893130
Cleaned Length: 38474840
Total Words Removed: 24418290
```

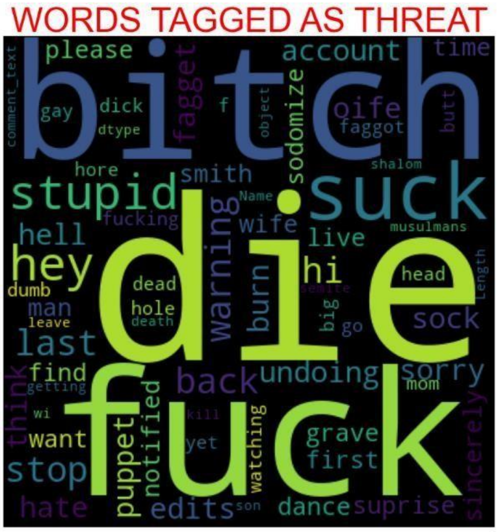
- **Length of comment_text data, before and after:**

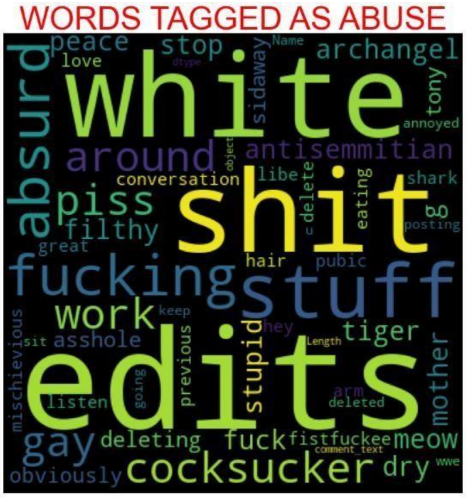
	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	nuetral	length_before_cleaning	len_after_cleaning
0	explanation edits made username hardcore metal...	0	0	0	0	0	0	1	264	156
1	aww match background colour seemingly stuck th...	0	0	0	0	0	0	1	112	67
2	hey man really trying edit war guy constantly ...	0	0	0	0	0	0	1	233	141
3	make real suggestion improvement wondered sect...	0	0	0	0	0	0	1	622	364
4	sir hero chance remember page	0	0	0	0	0	0	1	67	29

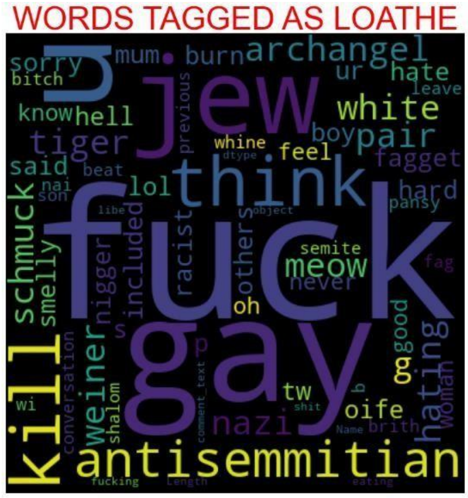
Similar step was used on test data. The dataset was processed using the function created.

- **Plotting wordcloud for each feature:**









From the above wordclouds, we can see that the large texts have more weightage in their respective type of comments whereas small texts have the lesser weightages

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

- For doing this project, the hardware used is a laptop with high end specification and a stable internet connection. While coming to software part, I had used anaconda navigator and in that I have used **Jupyter notebook** to do my python programming and analysis.

- For using an CSV file, Microsoft excel is needed. In Jupyter notebook, I had used lots of python libraries to carry out this project and I have mentioned below with proper justification:

```python
#Importing warning library to avoid any warnings
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#Importing Required libraries for NLP
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

#Importing the Oversampling library and Counter for handling imbalanced dataser
from collections import Counter
from imblearn.over_sampling import RandomOverSampler

#Libraries for model building
from sklearn.model_selection import train_test_split
#Importing required libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import hamming_loss, log_loss

#Saving the model
import pickle
```

MODEL/S DEVELOPMENT AND EVALUATION

- **Separating independent and dependent features and converting the data into number features using Vectorizer**

```python
#Converting the features into number vectors
tf_vec = TfidfVectorizer(max_features = 15000, stop_words='english')
```

```python
#Let's Separate the input and output variables represented by X and y respectively in train data and convert them
X = tf_vec.fit_transform(df_train['comment_text'])
```

```python
y=df_train['label']
```

```python
print(X.shape,'\t\t', y.shape)    #Checking the shape of the data
```

```
(159571, 15000)                  (159571,)
```

```python
#Doing the above process for test data
test_vec = tf_vec.fit_transform(df_test['comment_text'])
test_vec
```

```
<153164x15000 sparse matrix of type '<class 'numpy.float64'>'
	with 2870432 stored elements in Compressed Sparse Row format>
```

```python
test_vec.shape
```

```
(153164, 15000)
```

- **Splitting training and testing data, along with handling imbalanced dataset using RandomOverSampler**

```
#Splitting the training and testing data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)
```

```
#Checking the shape of x data
print(x_train.shape,'\t\t',x_test.shape)
```

```
(111699, 15000)                 (47872, 15000)
```

```
#Checking the shape of y data
print(y_train.shape,'\t',y_test.shape)
```

```
(111699,)        (47872,)
```

Handling the imbalanced data using oversampling technique

```
#Importing the Oversampling library and Counter
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
```

```
#We are trying to increase the points of minimum label data
os = RandomOverSampler(0.75)
x_train_os,y_train_os = os.fit_resample(x_train,y_train)
print("The number of classes before fit {}".format(Counter(y_train)))
print("The number of classes after fit {}".format(Counter(y_train_os)))
```

```
The number of classes before fit Counter({0: 100342, 1: 11357})
The number of classes after fit Counter({0: 100342, 1: 75256})
```

After oversampling, we can see that the dataset is balanced, by increasing the weightage of the lowest kind with sampling 75% of the highest weightage data to it.

- **Building the model**

```
#Initializing the instance of the model
LR=LogisticRegression()
mnb=MultinomialNB()
dtc=DecisionTreeClassifier()
knc=KNeighborsClassifier()
rfc=RandomForestClassifier()
abc=AdaBoostClassifier()
gbc=GradientBoostingClassifier()
xgb=XGBClassifier()
```

```
models= []
models.append(('Logistic Regression',LR))
models.append(('MultinomialNB',LR))
models.append(('DecisionTreeClassifier',dtc))
models.append(('KNeighborsClassifier',knc))
models.append(('RandomForestClassifier',rfc))
models.append(('AdaBoostClassifier',abc))
models.append(('GradientBoostingClassifier',gbc))
models.append(('XGBoostClassifier',xgb))
```

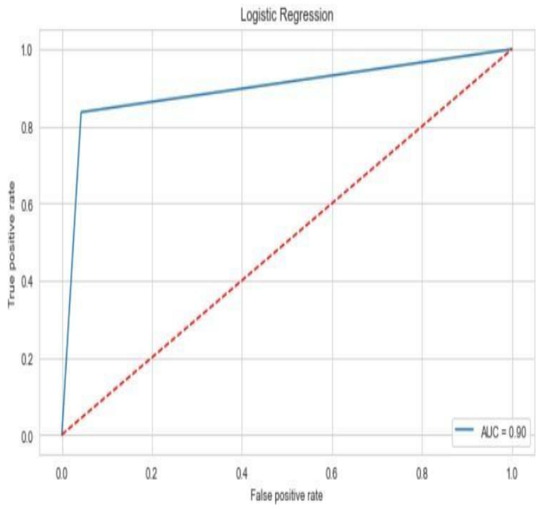
```python
#Making a for loop and calling the algorithm one by one and save data to respective model using append function
Model=[]
score=[]
cvs=[]
rocscore=[]
h_loss=[]
l_loss=[]
for name,model in models:
    print('******************************',name,'**************************')
    print('\n')
    Model.append(name)
    model.fit(x_train_os,y_train_os)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=accuracy_score(y_test,pre)
    print('accuracy_score: ',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,X,y,cv=5,scoring='accuracy').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    false_positive_rate,true_positive_rate,thresholds=roc_curve(y_test,pre)
    roc_auc= auc(false_positive_rate,true_positive_rate)
    print('roc_auc_score: ',roc_auc)
    rocscore.append(roc_auc*100)
    print('\n')
    hloss = hamming_loss(y_test, pre)
    print("Hamming_loss:", hloss)
    h_loss.append(hloss)
    print('\n')
    try :
        loss = log_loss(y_test, pre)
    except :
            loss = log_loss(y_test, pre.toarray())
    print("Log_loss :", loss)
    l_loss.append(loss)
    print('\n')
    print('Classification report:\n ')
    print(classification_report(y_test,pre))
    print('\n')
    print('Confusion matrix: \n')
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n')
    plt.figure(figsize=(10,50))
    plt.subplot(912)
    print('AUC_ROC curve:\n')
    plt.title(name)
    plt.plot(false_positive_rate,true_positive_rate, label='AUC = %0.2f'% roc_auc)
    plt.plot([0,1],[0,1],'r--')
    plt.legend(loc='lower right')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.show()

    print('\n\n\n')
```

After running the above for loops for the algorithms, the output will be as follows:

Logistic Regression

```
LogisticRegression()

accuracy_score:  0.9449364973262032

cross_val_score:  0.956094776134958

roc_auc_score:  0.896848218892634

Hamming_loss: 0.05506350267379679

Log_loss : 1.901856739560832

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97     43004
           1       0.69      0.84      0.76      4868

    accuracy                           0.94     47872
   macro avg       0.83      0.90      0.86     47872
weighted avg       0.95      0.94      0.95     47872


Confusion matrix:

[[41164  1840]
 [  796  4072]]
```

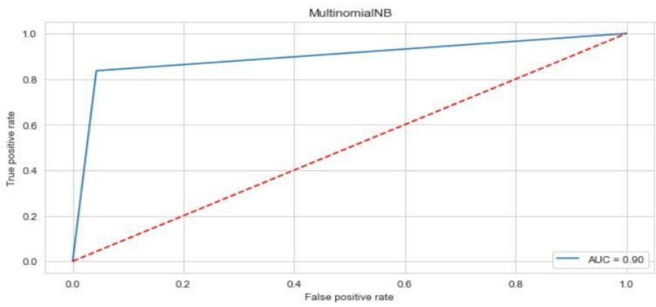
AUC_ROC curve:



MultinomialNB

```
LogisticRegression()

accuracy_score:  0.9449364973262032

cross_val_score:  0.956094776134958

roc_auc_score:  0.896848218892634

Hamming_loss: 0.05506350267379679

Log_loss : 1.901856739560832

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97     43004
           1       0.69      0.84      0.76      4868

    accuracy                           0.94     47872
   macro avg       0.83      0.90      0.86     47872
weighted avg       0.95      0.94      0.95     47872


Confusion matrix:

[[41164  1840]
 [  796  4072]]
```

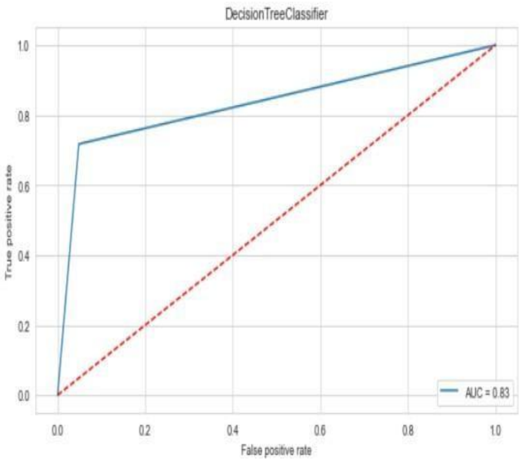
AUC_ROC curve:



DecisionTree

```
DecisionTreeClassifier()

accuracy_score:  0.9278283756684492

cross_val_score:  0.940791242671793

roc_auc_score:  0.8344966878164704

Hamming_loss: 0.0721716243315508

Log_loss : 2.4927543200090616

Classification report:

              precision    recall  f1-score   support

           0       0.97      0.95      0.96     43004
           1       0.63      0.72      0.67      4868

    accuracy                           0.93     47872
   macro avg       0.80      0.83      0.81     47872
weighted avg       0.93      0.93      0.93     47872


Confusion matrix:

[[40925  2079]
 [ 1376  3492]]
```

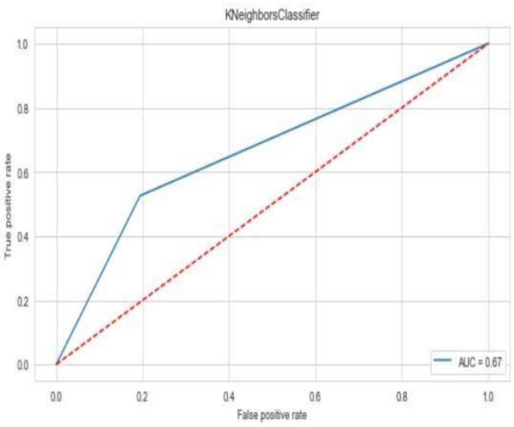
AUC_ROC curve:



KNeighbors

```
accuracy_score:  0.7774690842245989

cross_val_score:  0.9178610118092028

roc_auc_score:  0.6658247169990569

Hamming_loss: 0.22253091577540107

Log_loss : 7.68608490925679

Classification report:

              precision    recall  f1-score   support

           0       0.94      0.81      0.87     43004
           1       0.23      0.53      0.32      4868

    accuracy                           0.78     47872
   macro avg       0.59      0.67      0.60     47872
weighted avg       0.87      0.78      0.81     47872


Confusion matrix:

[[34660  8344]
 [ 2309  2559]]
```

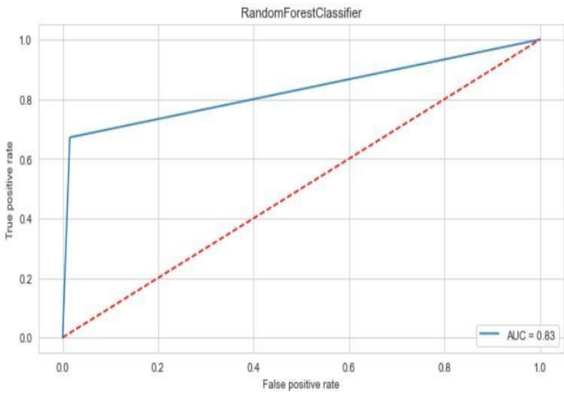
AUC_ROC curve:



RandomForest

```
RandomForestClassifier()

accuracy_score:  0.9530832219251337

cross_val_score:  0.9568154596427295

roc_auc_score:  0.8282416181575512

Hamming_loss: 0.04691677807486631

Log_loss : 1.6204589138224175

Classification report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.83      0.67      0.74      4868

    accuracy                           0.95     47872
   macro avg       0.90      0.83      0.86     47872
weighted avg       0.95      0.95      0.95     47872


Confusion matrix:

[[42357   647]
 [ 1599  3269]]
```

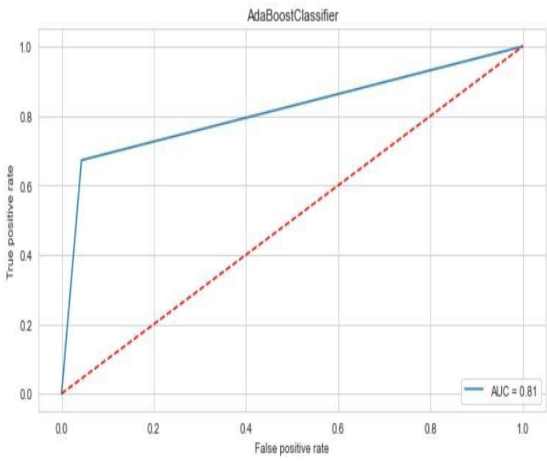
AUC_ROC curve:



Adaboost

```
AdaBoostClassifier()

accuracy_score:  0.9278910427807486

cross_val_score:  0.9459174938176664

roc_auc_score:  0.8145840057529954

Hamming_loss: 0.07210895721925134

Log_loss : 2.4905861666038516

Classification report:

              precision    recall  f1-score   support

           0       0.96      0.96      0.96     43004
           1       0.64      0.67      0.65      4868

    accuracy                           0.93     47872
   macro avg       0.80      0.81      0.81     47872
weighted avg       0.93      0.93      0.93     47872


Confusion matrix:

[[41147  1857]
 [ 1595  3273]]
```

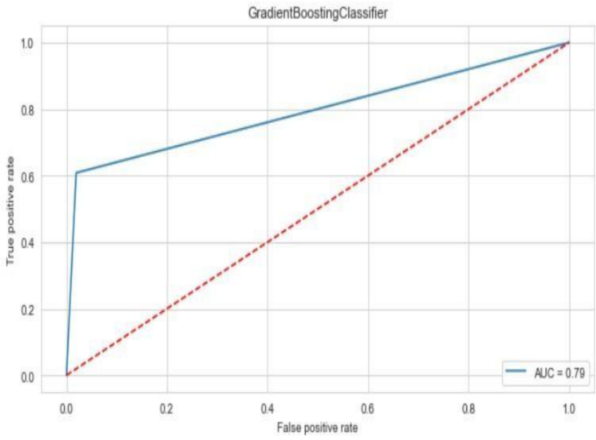
AUC_ROC curve:



GradientBoosting

```
GradientBoostingClassifier()

accuracy_score:  0.9428267045454546

cross_val_score:  0.9402711016469325

roc_auc_score:  0.7944787406602295

Hamming_loss: 0.057173295454545456

Log_loss : 1.9747095308058498

Classification report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.78      0.61      0.68      4868

    accuracy                           0.94     47872
   macro avg       0.87      0.79      0.83     47872
weighted avg       0.94      0.94      0.94     47872


Confusion matrix:

[[42174   830]
 [ 1907  2961]]
```

AUC_ROC curve:



XGBoost

```
accuracy_score:  0.9490516377005348

[15:35:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[15:36:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[15:38:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[15:39:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[15:40:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
cross_val_score:  0.9539201927128083


roc_auc_score:  0.8547804251569878


Hamming_loss: 0.05094836229946524


Log_loss : 1.7597134016088005
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43004
           1       0.76      0.74      0.75      4868

    accuracy                           0.95     47872
   macro avg       0.86      0.85      0.86     47872
weighted avg       0.95      0.95      0.95     47872


Confusion matrix:
[[41848  1156]
 [ 1283  3585]]

AUC_ROC curve:
```


```python
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'Accuracy_score': score,'Cross_val_score':cvs,'roc_auc_score':rocscore,
                     'Hamming_loss':h_loss, 'Log_loss':l_loss})
result
```

	Model	Accuracy_score	Cross_val_score	roc_auc_score	Hamming_loss	Log_loss
0	Logistic Regression	94.493650	95.609478	89.684822	0.055064	1.901857
1	MultinomialNB	94.493650	95.609478	89.684822	0.055064	1.901857
2	DecisionTreeClassifier	92.782838	94.079124	83.449669	0.072172	2.492754
3	KNeighborsClassifier	77.746908	91.786101	66.582472	0.222531	7.686085
4	RandomForestClassifier	95.308322	95.681546	82.824162	0.046917	1.620459
5	AdaBoostClassifier	92.789104	94.591749	81.458401	0.072109	2.490586
6	GradientBoostingClassifier	94.282670	94.027110	79.447874	0.057173	1.974710
7	XGBoostClassifier	94.905164	95.392019	85.478043	0.050948	1.759713

After running the for loop of classification algorithms and the required metrics, we can see that the best 2 performing algorithms are RandomForestClassifier and XGBoostClassifier because the loss values are less and their scores are the best among all. Now, we will try Hyperparameter Tuning to find out the best parameters and using them to improve the scores and metrics values.

- **Hyperparameter Tuning**

If we run GridSearchCV and RandomSearchCV, it takes more than 2 hours to run the code as the dataset is huge and the best params are not obtained from it due to more computational power requirement. The AUC Score, f1-score and recall value is high when we use randomforest with over sampling data. So, we choose RandomForestClassifier model with over sampled data as our best model among all models.

```
rfc = RandomForestClassifier()
rfc.fit(x_train_os,y_train_os)
```

```
RandomForestClassifier()
```

```
pred=rfc.predict(x_test)
print('Accuracy score: ',accuracy_score(y_test,pre)*100)
print('Cross validation score: ',cross_val_score(rfc,X,y,cv=5,scoring='accuracy').mean()*100)
false_positive_rate,true_positive_rate,thresholds=roc_curve(y_test,pre)
roc_auc= auc(false_positive_rate,true_positive_rate)
print('roc_auc_score: ',roc_auc)
hloss = hamming_loss(y_test, pre)
print("Hamming_loss:", hloss)
loss = log_loss(y_test, pre)
print("Log loss:", loss)
print('Classification report: \n')
print(classification_report(y_test,pre))
print('Confusion matrix: \n')
print(confusion_matrix(y_test,pre))
```

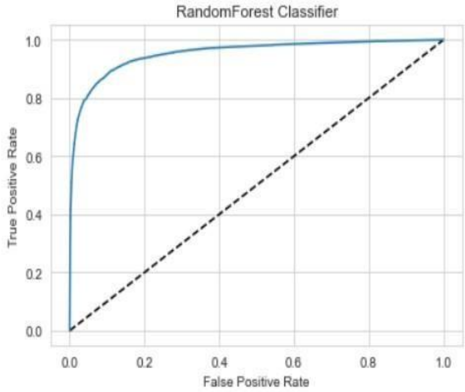
```
Accuracy score:  94.90516377005348
Cross validation score:  95.66838520306182
roc_auc_score:  0.8547804251569878
Hamming_loss: 0.05094836229946524
Log loss: 1.7597134016088005
Classification report:

              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43004
           1       0.76      0.74      0.75      4868

    accuracy                           0.95     47872
   macro avg       0.86      0.85      0.86     47872
weighted avg       0.95      0.95      0.95     47872
```

Confusion matrix:

[[41848 1156]
 [1283 3585]]

After finding the metrics values and other required scores, we will plot the auc roc curve with the help of auc score obtained with the help of the fitted model above:

```
#AUC_ROC Curve of Randomforest Classifier with oversampled data
y_pred_prob=rfc.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label='RandomForest Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RandomForest Classifier')
plt.show()

auc_score=roc_auc_score(y_test,rfc.predict(x_test))
print(auc_score)
```




```
0.830404398757655
```

- **Finalizing the model**

We will final the model by predicting the values and saving the model in a pickle file, which will be used for prediction of test data

```
rfc_prediction=rfc.predict(X)
#Making a dataframe of predictions
malignant_prediction=pd.DataFrame({'Predictions':rfc_prediction})
malignant_prediction
```

	Predictions
0	0
1	0
2	0
3	0
4	0
...	...
159566	0
159567	0
159568	0
159569	0
159570	0

159571 rows × 1 columns

```
#Saving the model
import pickle
filename='MalignantCommentsClassifier_Project.pkl'   #Specifying the filename
pickle.dump(rfc,open(filename,'wb'))
```

- **Predicting using test data**

```
#Checking our vectorized test data
test_vec
```

```
<153164x15000 sparse matrix of type '<class 'numpy.float64'>'
	with 2870432 stored elements in Compressed Sparse Row format>
```

```
#Loading the model
fitted_model=pickle.load(open('MalignantCommentsClassifier_Project.pkl','rb'))
fitted_model
```

```
RandomForestClassifier()
```

```
#Predictions
test_prediction=rfc.predict(test_vec)
test_df=pd.DataFrame({'Predictions':test_prediction})
test_df
```

	Predictions
0	1
1	0
2	0
3	0
4	0
...	...
153159	0
153160	0
153161	0
153162	0
153163	0

153164 rows × 1 columns

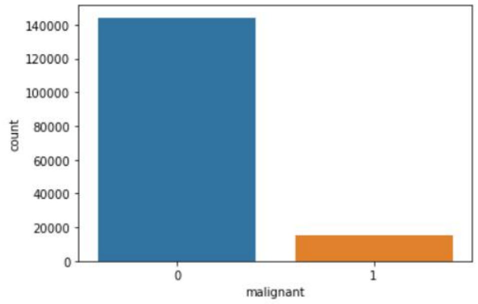
After predicting using test data, we will store the results in a csv file.

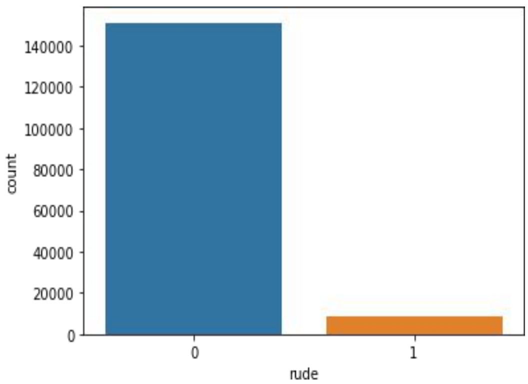
DATA VISUALIZATION

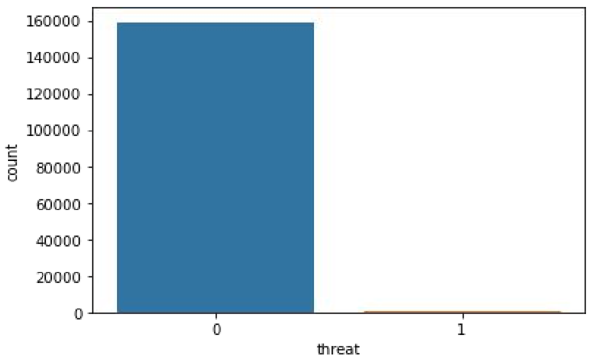
The use of tables, graphs, and charts play a vital role in presenting the data being used to draw these conclusions. Thus, data visualization is the best way to explore the data as it allows in-depth analysis.

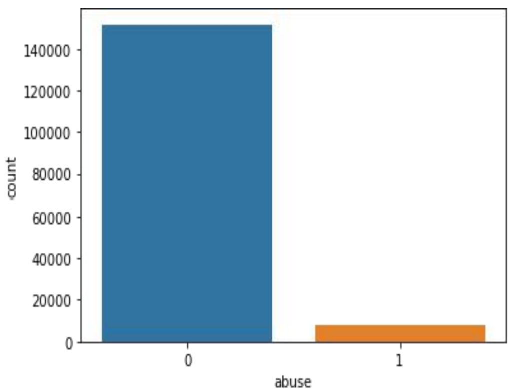
- **Plotting countplot for all features**

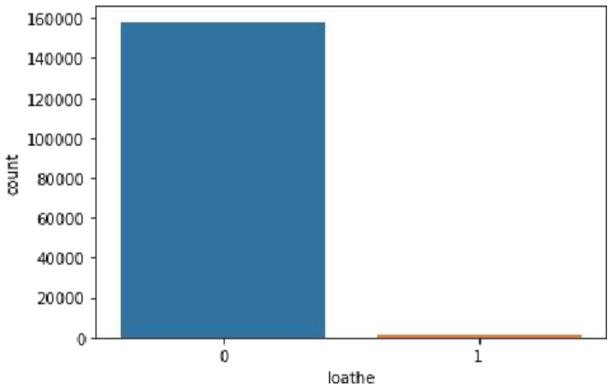
```
#Plotting countplot for all the features
categories=df_train.columns[2:]
for col in categories:
    sns.countplot(df_train[col])
    plt.show()
```





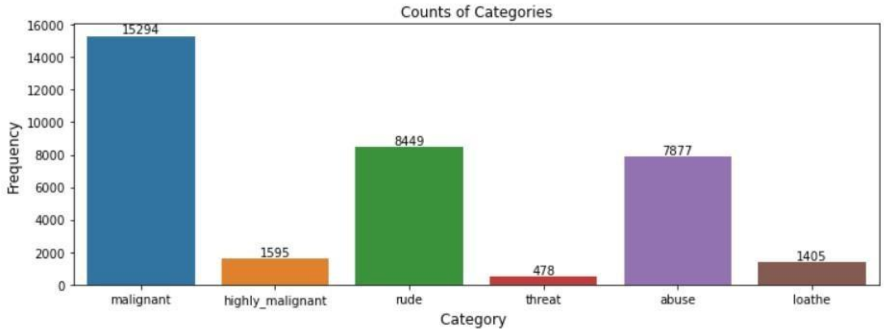






- **Plotting the counts for each category**

```
#Plotting the counts of each category
plt.figure(figsize=(12,4))
ax = sns.barplot(counts.index, counts.values)
plt.title("Counts of Categories")
plt.ylabel('Frequency', fontsize=12)
plt.xlabel('Category ', fontsize=12)
rects = ax.patches
labels = counts.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()
```

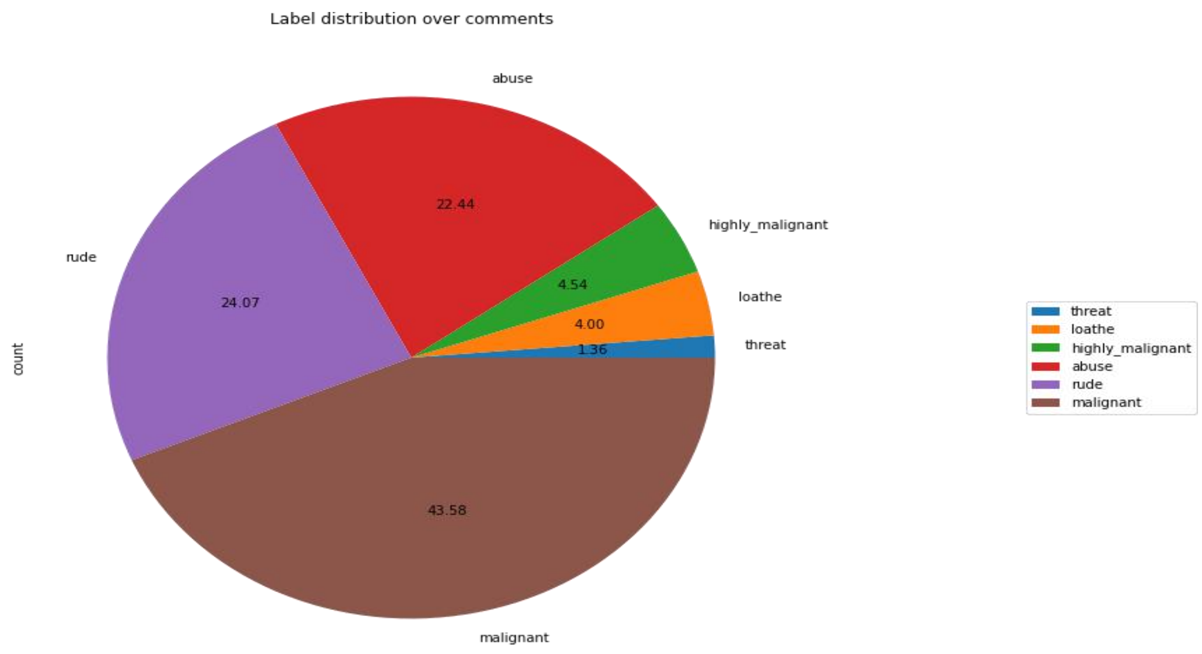


Malignant comments are the highest among all whereas threat comments are very less. Rude and abuse comments are also present more

- **Plotting pie chart plot**

```
#Visualizing the label distribution of comments using pie chart
comments_labels = ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']
df_distribution = df_train[comments_labels].sum()\
                            .to_frame()\
                            .rename(columns={0: 'count'})\
                            .sort_values('count')

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

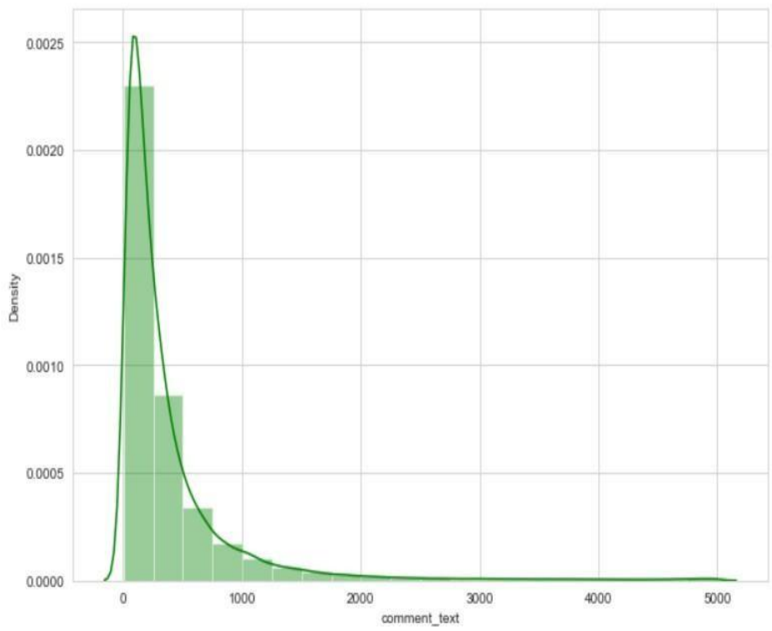
```
<matplotlib.legend.Legend at 0x1efd44e7ee0>
```



- **Distribution of comments length**

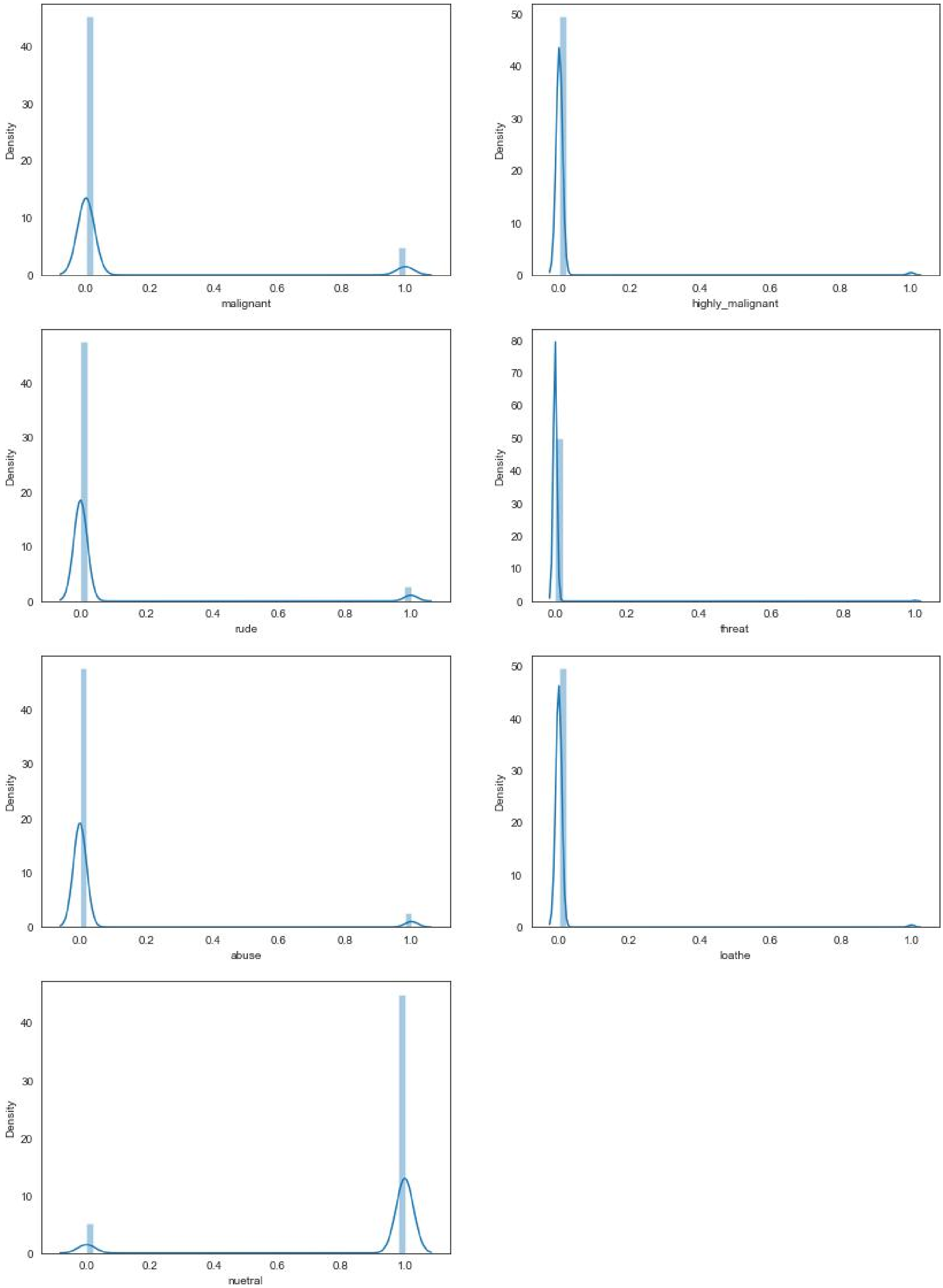
```
#Distribution of comments length
sns.set_style('whitegrid')
plt.figure(figsize=(10,7))
comment_len = df_train.comment_text.str.len()
sns.distplot(comment_len, bins=20, color = 'green')
```

```
<AxesSubplot:xlabel='comment_text', ylabel='Density'>
```



Above is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words. Majority of the comments are of length 500, where maximum length is 5000 and minimum length is 5. Median length being 250.

- **Plotting distribution plot for all features**



CONCLUSION

Key Findings and Conclusions of the Study

-> After the completion of this project, we got an insight of how to preprocess the data, analysing the data and building a model.

-> First, we imported both training and testing data, which had nearly 150000+ records.

-> We did all the required pre-processing steps like checking null values, datatypes check, dropping unnecessary columns, etc.

-> We used the training data for doing Exploratory Data Analysis using various plots and recorded the observations.

-> While observing the results, we found that the dataset was in highly imbalanced side and we need to handle it, in order to avoid overfitting problem.

-> Using NLP, we pre-processed the comment text and did other steps.

-> As the problem was a multi-class classifier, we took a new feature known as label and combined the comment_labels output together using sum() and then stored in that feature. For a binary classification problem, we scaled the data accordingly.

-> After applying Tf-idf Vectoriser, we used an oversampling technique called RandomOverSampler for handling the imbalanced data. There, we took 75% of the high points data and sampled it to the low points data so that both weights could be balanced equally and we could get proper result.

-> Then, we split the data using train_test_split and then we started the model building process by running as many algorithms in a for loop, with difference metrics like cross_val_score, confusion matrix, auc_score, log loss, hamming loss, etc.

-> We found that RandomForestClassifier and XGBoostClassifier were performing well. The next step was to perform hyperparameter tuning technique to these models for finding out the best parameters and trying to improve our scores.

-> The major problem with this dataset occurred in this step. It took me nearly 2 hrs to run the code for finding out the best parameters itself as the dataset is large and more computational power was required. Even though we found the best algorithms, it took me 2 hrs to get the results.

-> Therefore, without hyperparameter tuning, we finalized RandomForest as the best performing algorithm by predicting the outputs, saving the model and storing the results in a csv file

-> Then, by using the model we got, another set of predictions were done by using the test data and the results were stored in a separate csv file.

Problems faced while working in this project:

- More computational power was required as it took more than 2 hours
- Imbalanced dataset and bad comment texts
- Good parameters could not be obtained using hyperparameter tuning as time was consumed more

Areas of improvement:

- Could be provided with a good dataset which does not take more time.
- Less time complexity
- Providing a proper balanced dataset with less errors.

Learning Outcomes of the Study in respect of Data Science

Through this project we were able to learn various Natural language processing techniques like lemmatization, stemming, removal of stopwords. We were also able to learn to convert strings into vectors through hash vectorizer. In this project we applied different evaluation metrics like log loss, hamming loss besides accuracy.

My point of view from my project is that we need to use proper words which are respectful and also avoid using abusive, vulgar and worst words in social media. It can cause many problems which could affect our lives. Try to be polite, calm and composed while handling stress and negativity and one of the best solutions is to avoid it and overcoming in a positive manner.