MALIGNANT COMMENTS CLASSIFIER PROJECT

Submitted by:

Roshan Kumar Verma

ACKNOWLEDGMENT

This Project 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 would like to thank 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 would like to express my gratitude towards my parents & my SME of Flip-Robo Mohd. Kashif for their kind co-operation and encouragement which help
Me in completion of this project.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

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.

Conceptual Background of the Domain Problem

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.

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

Mathematical/ Analytical Modeling

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.

Malignant: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.

Highly Malignant: It denotes comments that are highly malignant and hurtful.

Rude: It denotes comments that are very rude and offensive.

Threat: It contains indication of the comments that are giving any threat to someone.

Abuse: It is for comments that are abusive in nature.

Loathe: It describes the comments which are hateful and loathing in nature.

ID: It includes unique Ids associated with each comment text given.

Comment text: This column contains the comments extracted from various social media platforms.

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:

In [2]:
```
#Loading train data
df_train=pd.read_csv('train.csv')  #Path location of the dataset
df_train.head()  #Checking out the top 5 rows of the dataset
```

Out[2]:

	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

In [10]:
```
#Loading test data
df_test=pd.read_csv('test.csv')  #Path location of the dataset
df_test.head()  #Checking out the top 5 rows of the dataset
```

Out[10]:

	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 Done

Checking the value counts of the features

Exploratory Data Analysis

```
In [4]: 1 df_train['malignant'].value_counts()
```

```
Out[4]: 0    144277
        1     15294
        Name: malignant, dtype: int64
```

```
In [5]: 1 df_train['highly_malignant'].value_counts()
```

```
Out[5]: 0    157976
        1      1595
        Name: highly_malignant, dtype: int64
```

```
In [6]: 1 df_train['rude'].value_counts()
```

```
Out[6]: 0    151122
        1      8449
        Name: rude, dtype: int64
```

```
In [7]: 1 df_train['threat'].value_counts()
```

```
Out[7]: 0    159093
        1       478
        Name: threat, dtype: int64
```

```
In [8]: 1 df_train['abuse'].value_counts()
```

```
Out[8]: 0    151694
        1      7877
        Name: abuse, dtype: int64
```

```
In [9]: 1 df_train['loathe'].value_counts()
```

```
Out[9]: 0    158166
        1      1405
        Name: loathe, dtype: int64
```

Checking for null values

In [12]:
```
#Checking the datatype of the training dataset
df_train.dtypes
```

Out[12]:
```
id                object
comment_text      object
malignant          int64
highly_malignant   int64
rude               int64
threat             int64
abuse              int64
loathe             int64
dtype: object
```

id and comment_text are of object datatype, whereas the other datas are of int64 type.

In [13]:
```
df_train.isnull().sum()   #Checking for null values in the train dataset
```

Out[13]:
```
id                0
comment_text      0
malignant         0
highly_malignant  0
rude              0
threat            0
abuse             0
loathe            0
dtype: int64
```

In [14]:
```
#Checking the datatype of the testing dataset
df_test.dtypes
```

Out[14]:
```
id              object
comment_text    object
dtype: object
```

In [15]:
```
df_test.isnull().sum()   #Checking for null values in the test dataset
```

Out[15]:
```
id              0
comment_text    0
dtype: int64
```

Checking statistical summary of the dataset

In [23]:
```
1 df_train.describe()    #Statistical summary of the dataset
```

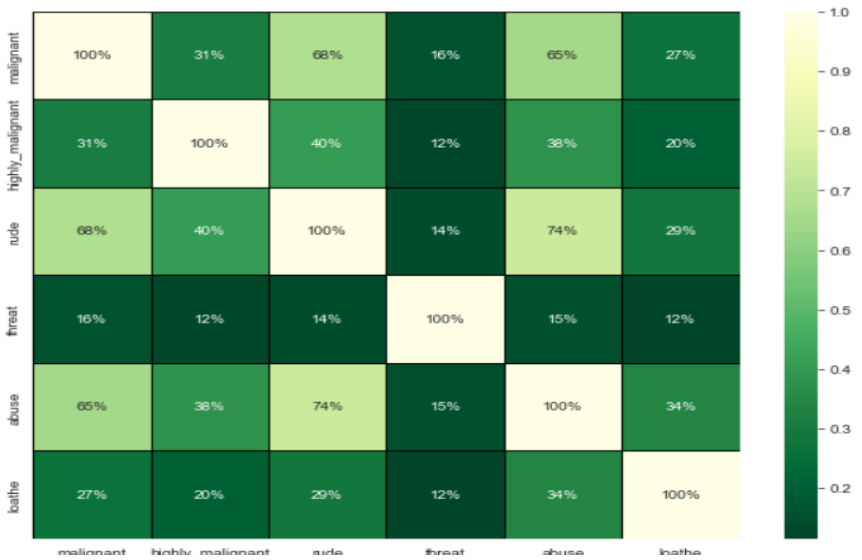
Out[23]:

	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

In [25]:
```
1 #Plotting heatmap for visualizing the correlation
2 plt.figure(figsize=(10,8))
3 sns.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',cmap='YlGn_r',annot=True)
4 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

```
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)
```

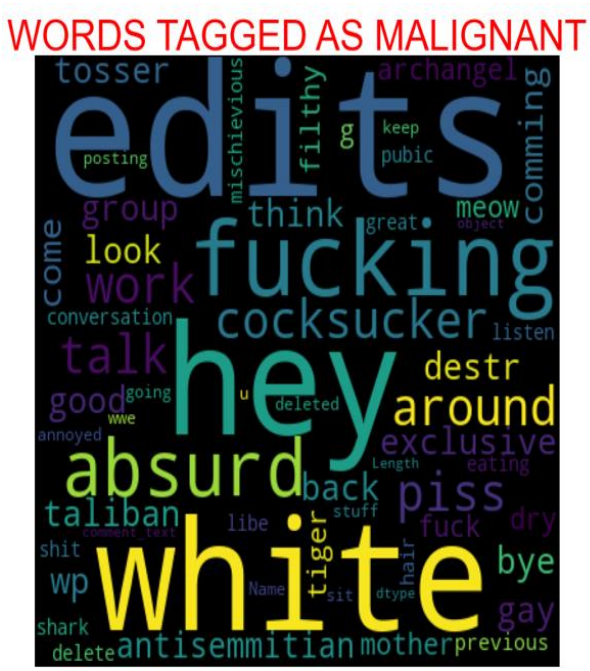
All the text cleaning or the above steps are performed by defining a function and applying the same using apply() to the comment_text column of the train dataset.

Data Inputs- Logic- Output Relationships

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.

Plotting wordcloud for each feature :

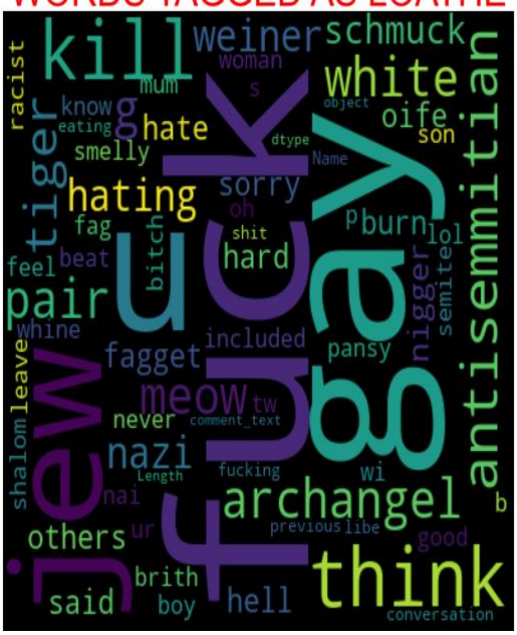


WORDS TAGGED AS RUDE

WORDS TAGGED AS THREAT

WORDS TAGGED AS ABUSE

WORDS TAGGED AS LOATHE



Hardware and Software Requirements and Tools Used

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

Model/s Development and Evaluation

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

Separating independent and dependent variables

In [70]:
```
#Converting the features into number vectors
tf_vec = TfidfVectorizer(max_features = 15000, stop_words='english')
```

In [71]:
```
#Let's Separate the input and output variables represented by X and y respectively in train data
X = tf_vec.fit_transform(df_train['comment_text'])
```

In [72]:
```
y=df_train['label']
```

In [73]:
```
print(X.shape,'\t\t', y.shape)    #Checking the shape of the data
```

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

In [74]:
```
#Doing the above process for test data
test_vec = tf_vec.fit_transform(df_test['comment_text'])
test_vec
```

Out[74]:
```
<153164x15000 sparse matrix of type '<class 'numpy.float64'>'
        with 2870432 stored elements in Compressed Sparse Row format>
```

In [75]:
```
test_vec.shape
```

Out[75]:
```
(153164, 15000)
```

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

In [76]:
```
#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)
```

In [77]:
```
#Checking the shape of x data
print(x_train.shape,'\t\t',x_test.shape)
```

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

In [78]:
```
#Checking the shape of y data
print(y_train.shape,'\t',y_test.shape)
```

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

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()
```

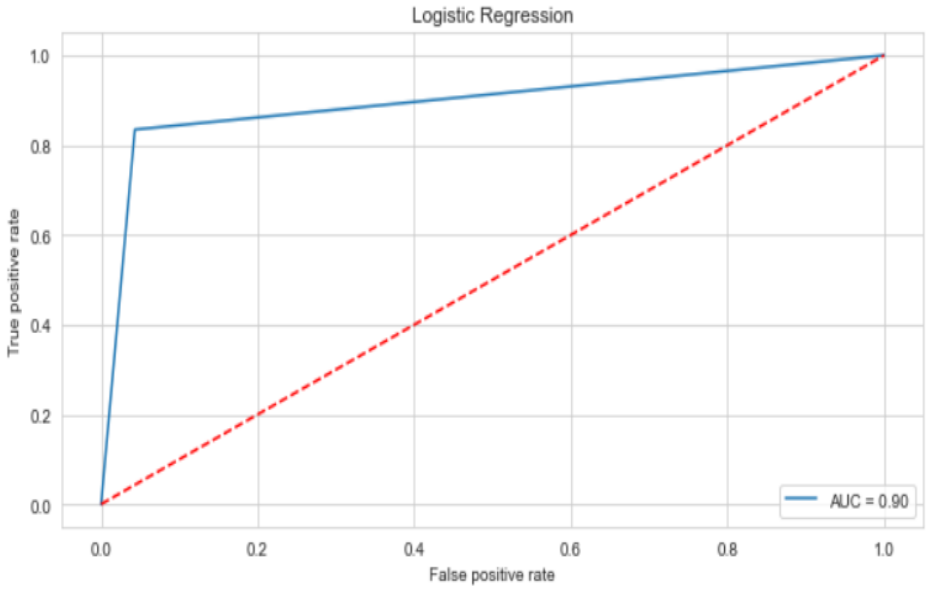
```
models= []
models.append(('Logistic Regression',LR))
models.append(('MultinomialNB',LR))
models.append(('DecisionTreeClassifier',dtc))
models.append(('KNeighborsClassifier',knc))
models.append(('RandomForestClassifier',rfc))
models.append(('AdaBoostClassifier',abc))
models.append(('GradientBoostingClassifier',gbc))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

```
#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')
```

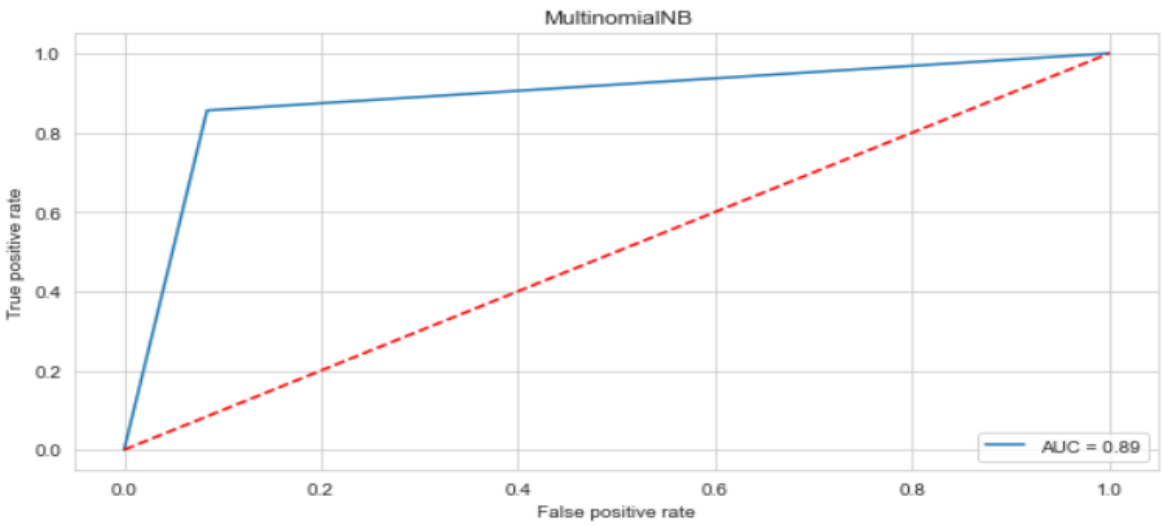
Logistic Regression

```
LogisticRegression()

accuracy_score:  0.9444978275401069

cross_val_score:  0.956094776134958

roc_auc_score:  0.895784292714893

Hamming_loss: 0.05550217245989305

Log_loss : 1.9170080576478616

Classification report:

              precision    recall  f1-score   support

           0       0.98      0.96      0.97     43004
           1       0.69      0.83      0.75      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:

[[41152  1852]
 [  805  4063]]


 AUC_ROC curve:
```



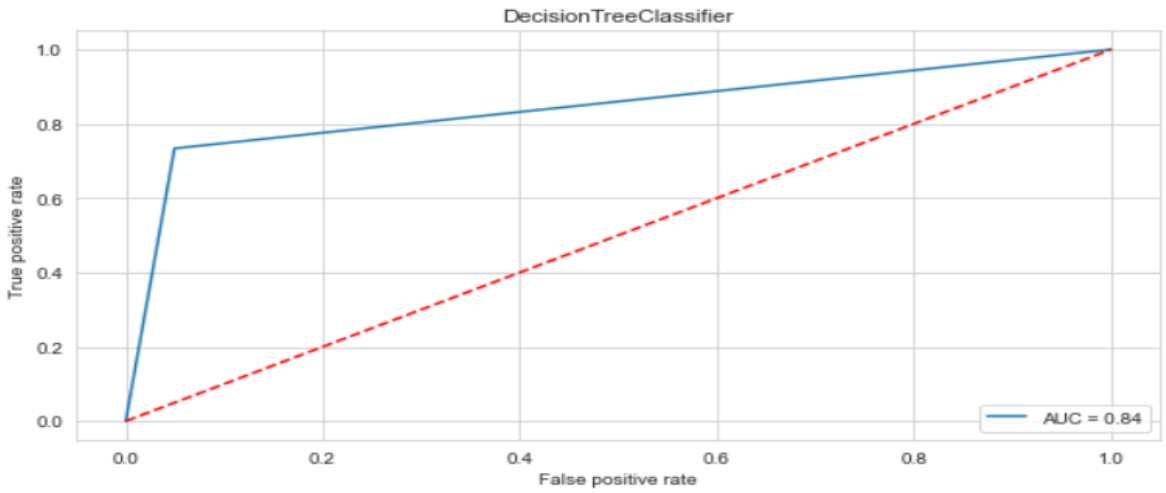
MultinomialNB

```
MultinomialNB()

accuracy_score:  0.9098011363636364

cross_val_score:  0.9466005743090046

roc_auc_score:  0.8859449507935933

Hamming_loss: 0.09019886363636363

Log_loss : 3.115418796313698

Classification report:

              precision    recall  f1-score   support

           0       0.98      0.92      0.95     43004
           1       0.54      0.86      0.66      4868

    accuracy                           0.91     47872
   macro avg       0.76      0.89      0.80     47872
weighted avg       0.94      0.91      0.92     47872


Confusion matrix:

[[39387  3617]
 [  701  4167]]

AUC_ROC curve:
```



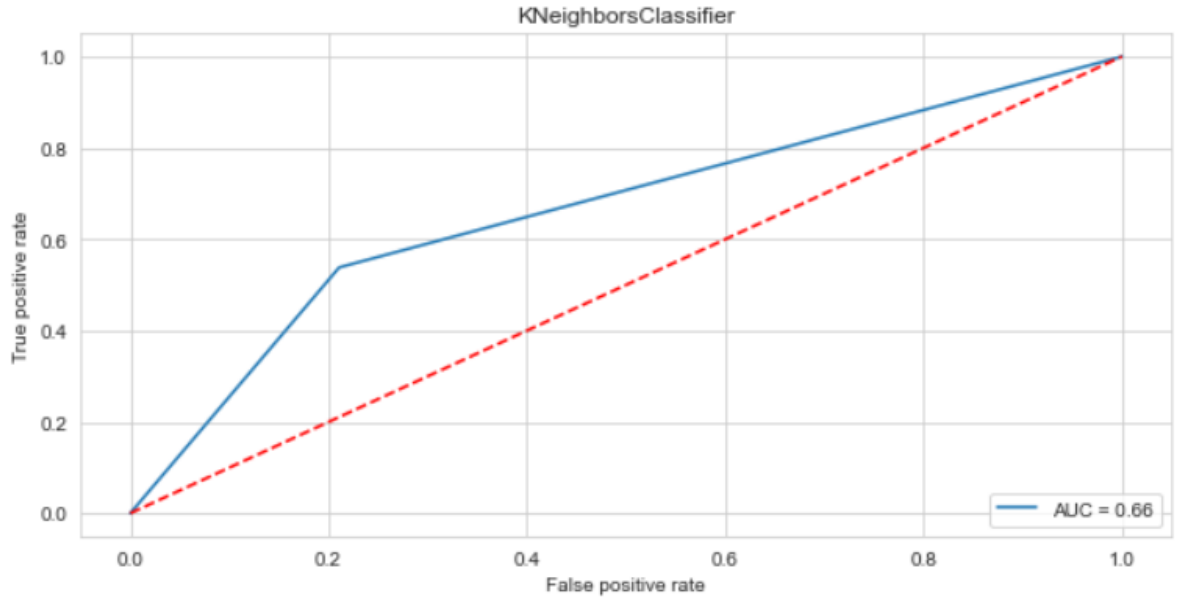
DecisionTreeClassifier

```
DecisionTreeClassifier()

accuracy_score:  0.9285386029411765

cross_val_score:  0.9408351060504716

roc_auc_score:  0.8421787807168881

Hamming_loss: 0.07146139705882353

Log_loss : 2.4682247073765176

Classification report:

              precision    recall  f1-score   support

           0       0.97      0.95      0.96     43004
           1       0.63      0.73      0.68      4868

    accuracy                           0.93     47872
   macro avg       0.80      0.84      0.82     47872
weighted avg       0.93      0.93      0.93     47872


Confusion matrix:

[[40879  2125]
 [ 1296  3572]]

AUC_ROC curve:
```



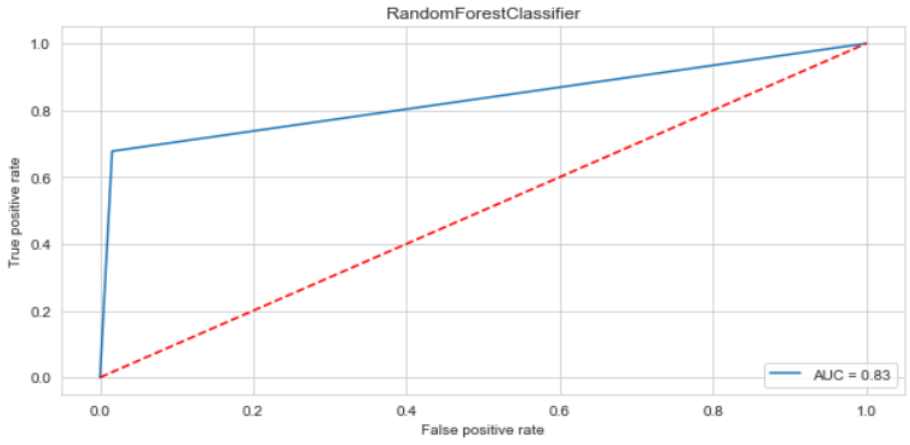
KNeighborsClassifier

```
KNeighborsClassifier()

accuracy_score:  0.7638494318181818

cross_val_score:  0.9178610118092028

roc_auc_score:  0.6638001972184736

Hamming_loss: 0.23615056818181818

Log_loss : 8.156502947409564

Classification report:

              precision    recall  f1-score   support

           0       0.94      0.79      0.86     43004
           1       0.22      0.54      0.32      4868

    accuracy                           0.76     47872
   macro avg       0.58      0.66      0.59     47872
weighted avg       0.87      0.76      0.80     47872

Confusion matrix:

[[33947  9057]
 [ 2248  2620]]

AUC_ROC curve:
```



RandomForestClassifier

```
****************************** RandomForestClassifier **************************

RandomForestClassifier()

accuracy_score:  0.9529787767379679

cross_val_score:  0.9568467893371316

roc_auc_score:  0.8307338573232415

Hamming_loss: 0.047021223262032084

Log_loss : 1.6240668739805422

Classification report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.83      0.68      0.75      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:

[[42324   680]
 [ 1571  3297]]

AUC_ROC curve:
```



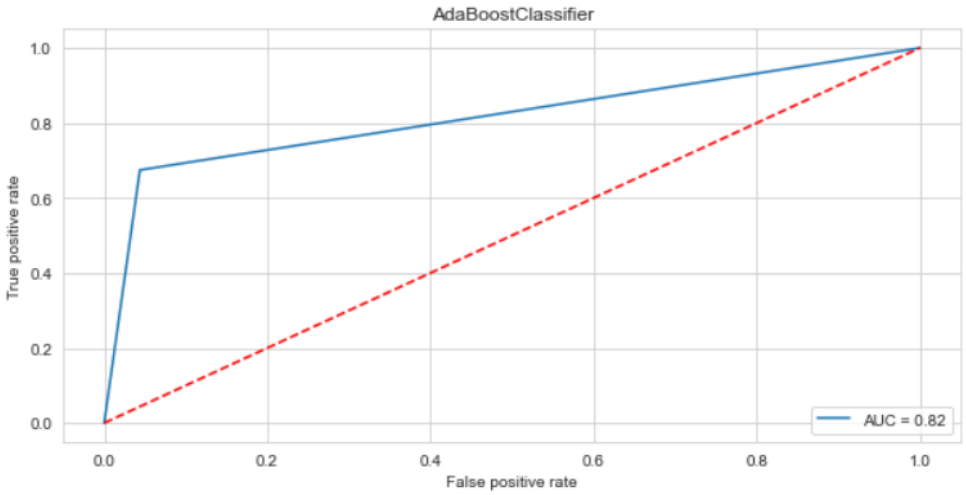
AdaBoostClassifier

```
AdaBoostClassifier()

accuracy_score:  0.9275985962566845

cross_val_score:  0.9459174938176664

roc_auc_score:  0.8153320778017861

Hamming_loss: 0.07240140374331551

Log_loss : 2.5006873125736684

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      0.82      0.81     47872
weighted avg       0.93      0.93      0.93     47872
```

Confusion matrix:

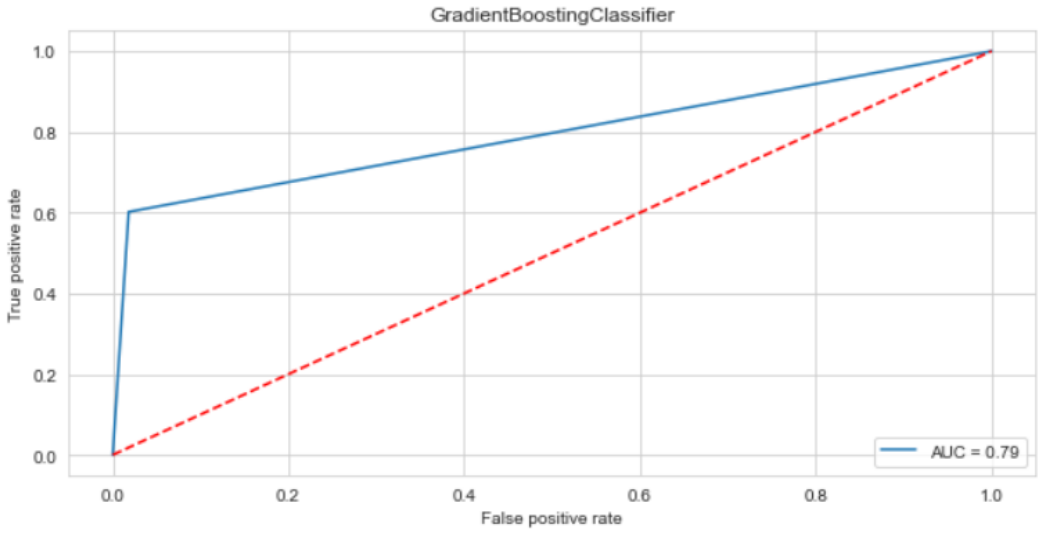
```
[[41123  1881]
 [ 1585  3283]]
```

AUC_ROC curve:



GradientBoostingClassifier

```
GradientBoostingClassifier()

accuracy_score:  0.9432235962566845

cross_val_score:  0.9402648351975152

roc_auc_score:  0.7918760227689354

Hamming_loss: 0.05677640374331551

Log_loss : 1.9610005415976528

Classification report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.79      0.60      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:

[[42224   780]
 [ 1938  2930]]

AUC_ROC curve:
```



XGBoostClassifier

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

```
accuracy_score:  0.94842496657754
```

```
cross_val_score:  0.9539201927128083
```

```
roc_auc_score:  0.8545227051550955
```

```
Hamming_loss: 0.05157503342245989
```

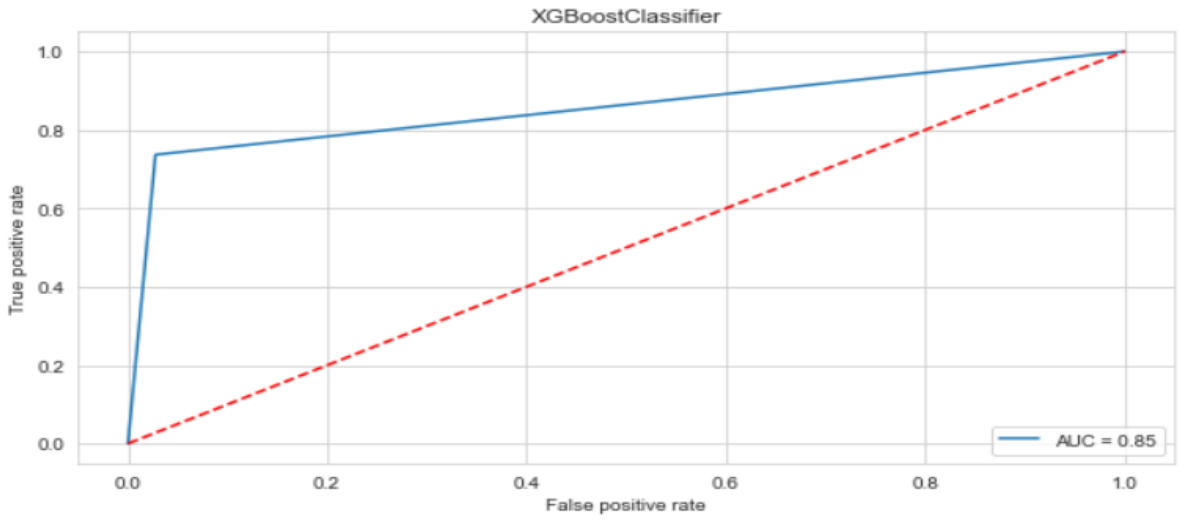
```
Log_loss : 1.7813583731865006
```

```
Classification report:
```

	precision	recall	f1-score	support
0	0.97	0.97	0.97	43004
1	0.75	0.74	0.74	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:

[[41817  1187]
 [ 1282  3586]]
```

```
AUC_ROC curve:
```



In [87]:

```
#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
```

Out[87]:

	Model	Accuracy_score	Cross_val_score	roc_auc_score	Hamming_loss	Log_loss
0	Logistic Regression	94.449783	95.609478	89.578429	0.055502	1.917008
1	MultinomialNB	90.980114	94.660057	88.594495	0.090199	3.115419
2	DecisionTreeClassifier	92.853860	94.083511	84.217878	0.071461	2.468225
3	KNeighborsClassifier	76.384943	91.786101	66.380020	0.236151	8.156503
4	RandomForestClassifier	95.297878	95.684679	83.073386	0.047021	1.624067
5	AdaBoostClassifier	92.759860	94.591749	81.533208	0.072401	2.500687
6	GradientBoostingClassifier	94.322360	94.026484	79.187602	0.056776	1.961001
7	XGBoostClassifier	94.842497	95.392019	85.452271	0.051575	1.781358

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.

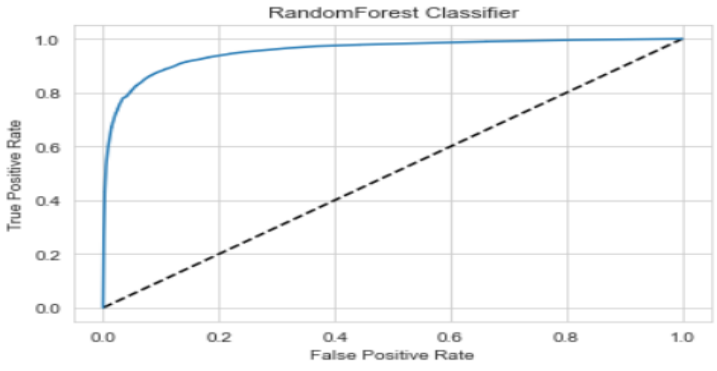
In [90]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train_os,y_train_os)
```

Out[90]:
```
RandomForestClassifier()
```

In [91]:
```
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,
false_positive_rate,true_positive_rate,thresholds=roc_curve(y_
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:  94.84249665775401
Cross validation score:  95.67590557071604
roc_auc_score:  0.8545227051550955
Hamming_loss: 0.05157503342245989
Log loss: 1.7813583731865006
Classification report:

              precision    recall  f1-score   support

           0       0.97      0.97      0.97     43004
           1       0.75      0.74      0.74      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:

[[41817  1187]
 [ 1282  3586]]
```

In [92]:
```
#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)
```




```
0.8325032079338017
```

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.

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.

In [93]:
```
rfc_prediction=rfc.predict(X)
#Making a dataframe of predictions
malignant_prediction=pd.DataFrame({'Predictions':r
malignant_prediction
```

Out[93]:

	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

In [94]:
```
#Saving the model
import pickle
filename='MalignantCommentsClassifier_Project.pkl'
pickle.dump(rfc,open(filename,'wb'))
```

Predicting using test data

In [95]:
```
#Checking our vectorized test data
test_vec
```

Out[95]:
```
<153164x15000 sparse matrix of type '<class 'numpy.float64'>'
        with 2870432 stored elements in Compressed Sparse Row format>
```

In [96]:
```
#Loading the model
fitted_model=pickle.load(open('MalignantCommentsClassifier_Project.pkl','rb'))
fitted_model
```

Out[96]:
```
RandomForestClassifier()
```

In [97]:
```
#Predictions
test_prediction=rfc.predict(test_vec)
test_df=pd.DataFrame({'Predictions':test_prediction})
test_df
```

Out[97]:

	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.

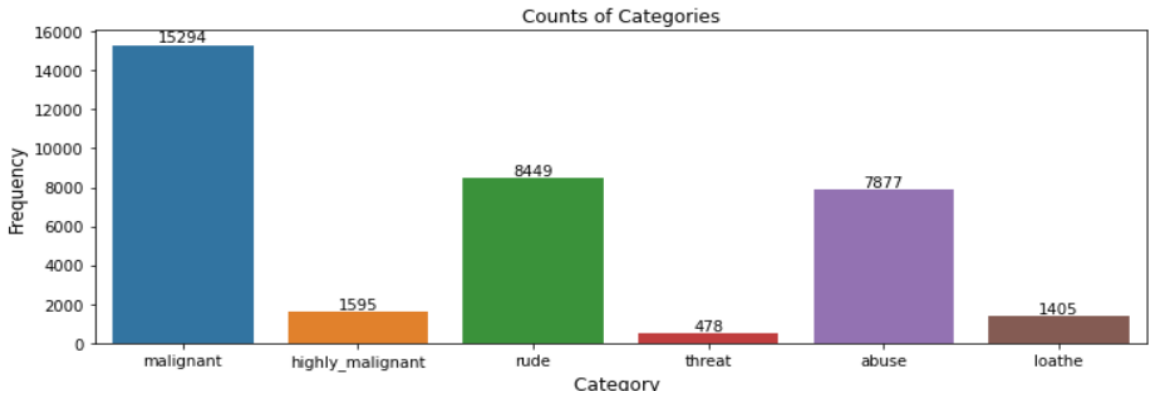
DATA VISUALIZATION

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 the counts for each category

In [19]:

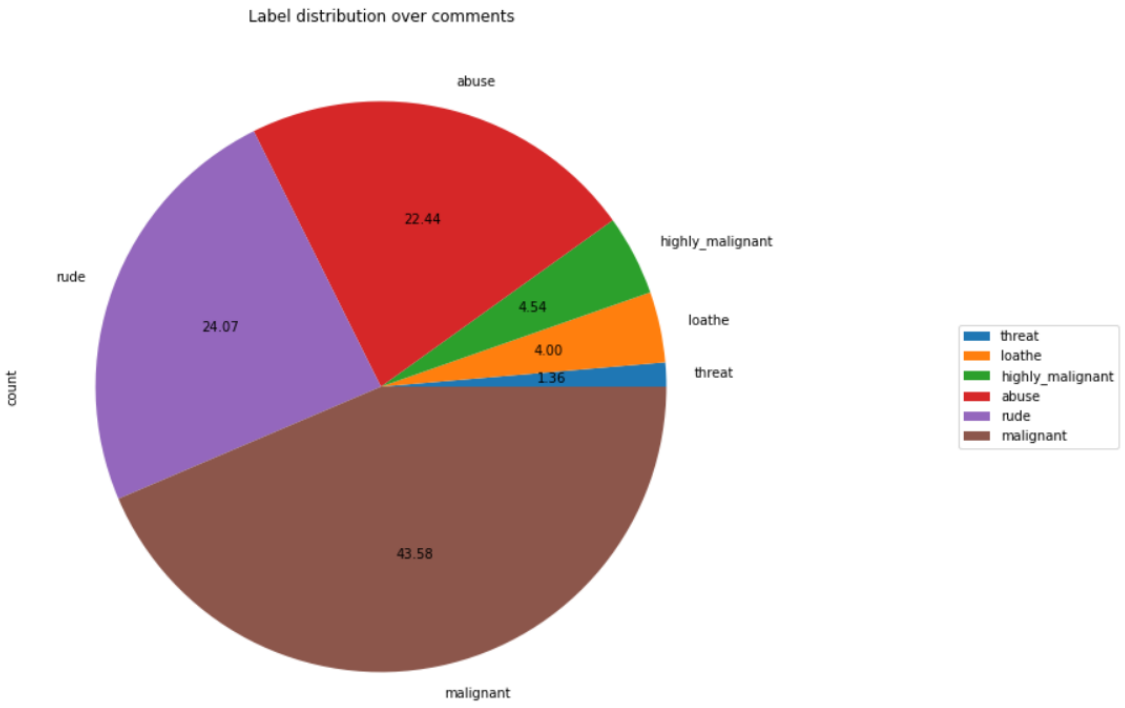
```
#Plotting the counts of each category
plt.figure(figsize=(12,4))
ax = sns.barplot(counts.index, counts.values)
plt.title("Counts of Categories")
plt.ylabel('Frequency', fontsize=12)
plt.xlabel('Category ', fontsize=12)
rects = ax.patches
labels = counts.values
for rect, label in zip(rects, labels):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

In [20]:
```
#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))
```

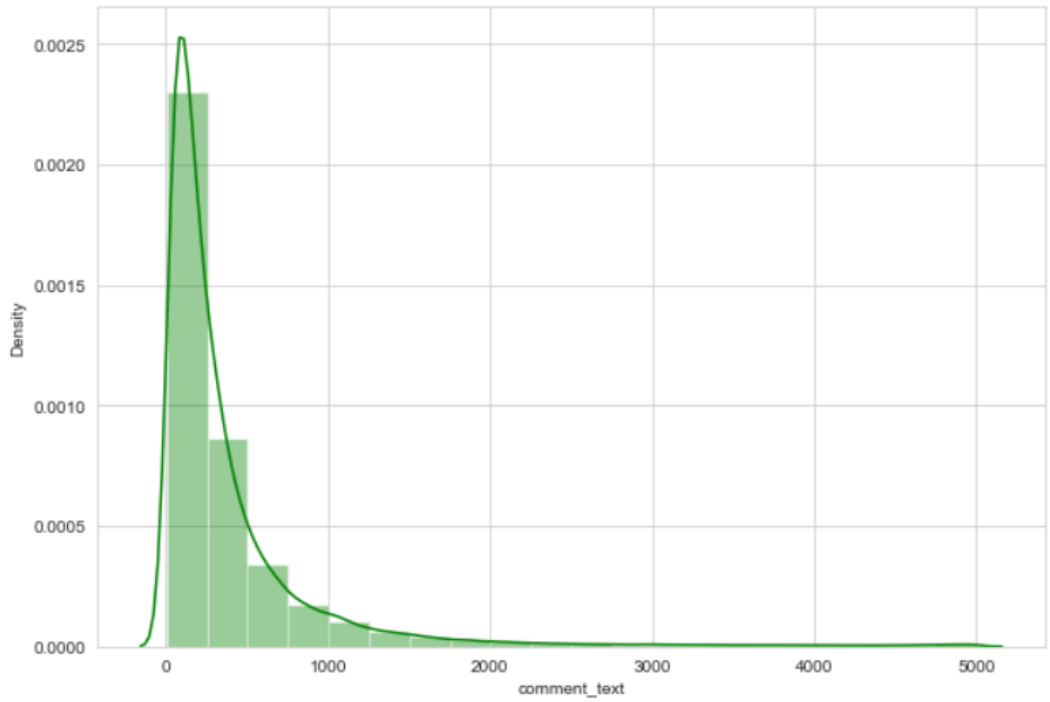


Distribution of comments length

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.

In [21]:
```
#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')
```

Out[21]: <AxesSubplot:xlabel='comment_text', ylabel='Density'>



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.