Malignant Comments Classifier Project

Submitted By:
SYED MUGERA BILAL
syedbilalarmaan@gmail.com

ACKNOWLEDGMENT

Working on this project ***"Malignant Comments Classifier Project ."*** Was a source of immense knowledge to me. We would like to express my sincere gratitude to FlipRobo Technologies for guidance and valuable support throughout this project work.

We acknowledge with a deep sense of gratitude, the encouragement and inspiration received from our SME/Mentor. We would also like to thank our parents for their love and support.

–SYED MUGERA BILAL

CONTENTS

About the project

- People use social media as a platform to express their opinions and views.
- However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users.
- Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms.
- 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.

MULTI-LABEL VS MULTI CLASS 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.
- The goal of the multi-label classification is to determine whether or not a comment is toxic or non-

toxic. If toxic, to determine what kind of toxicity the comment is, like-in, malignant, highly-malignant, threat, rude, abuse or loathe. Thus, a model needs to be created in order to differentiate between comments and its categories.

Data source and their format:

- 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 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.

Hardware, Software and Tools:

- ❖ For doing this project, we require laptop with high configuration and specification with a stable Internet connection.
- ❖ Microsoft office, Anaconda distribution as software.
- ❖ Python 3.x as programming language.
- ❖ Jupyter Notebook as Editor which is in Anaconda navigator.
- ❖ Some tools or libraries required like Numpy – used to numerical calculations. Pandas – used to data manipulation. Matplotlib and Seaborn – used to data visualization.

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

Training Dataset

```
In [4]: # Loading the train dataset
        df_train = pd.read_csv("train.csv")

In [5]: # Top 5 rows of train dataset
        df_train.head()
```

Data Analysis:

```
In [6]: # Checking the dimensions of the train dataset
        df_train.shape

Out[6]: (159571, 8)
```

- The train dataset has 159571 rows and 8 columns.

```
In [17]: # Checking the dimensions of the test dataset
         df_test.shape

Out[17]: (153164, 2)
```

- The test dataset has 154164 rows and 2 columns.

- Checking the value counts of the features.

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

Out[7]: 0    144277
        1     15294
        Name: malignant, dtype: int64

In [8]: df_train['highly_malignant'].value_counts()

Out[8]: 0    157976
        1      1595
        Name: highly_malignant, dtype: int64

In [9]: df_train['rude'].value_counts()

Out[9]: 0    151122
        1      8449
        Name: rude, dtype: int64

In [10]: df_train['threat'].value_counts()

Out[10]: 0    159093
         1       478
         Name: threat, dtype: int64

In [11]: df_train['abuse'].value_counts()

Out[11]: 0    151694
         1      7877
         Name: abuse, dtype: int64

In [12]: df_train['loathe'].value_counts()

Out[12]: 0    158166
         1      1405
         Name: loathe, dtype: int64
```

➢ Checking for null values.

```
In [14]: # Checking null values in the train dataset
         df_train.isnull().sum()

Out[14]: id                0
         comment_text      0
         malignant         0
         highly_malignant  0
         rude              0
         threat            0
         abuse             0
         loathe            0
         dtype: int64
```

```
In [19]: # Checking null values in the test dataset
         df_test.isnull().sum()

Out[19]: id              0
         comment_text    0
         dtype: int64
```

As we can see, there are no null values.

➢ Checking statistical summary of the dataset

```
In [27]: # Statistical summary of train dataset
         df_train.describe()
```

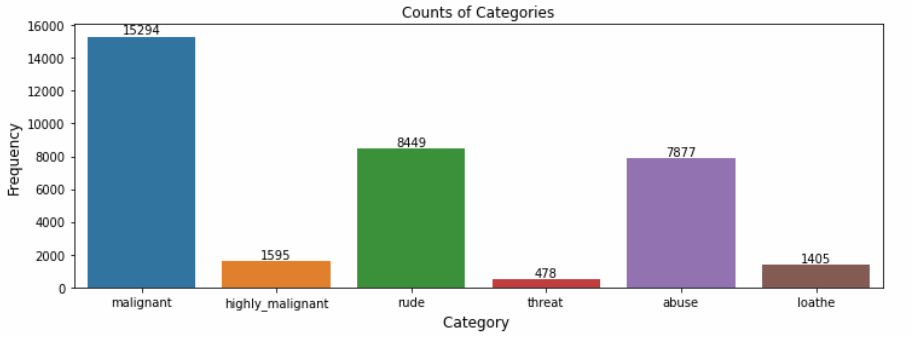
Out[27]:

	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.

Exploratory Data Analysis (EDA):

```
In [23]: # 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()
```

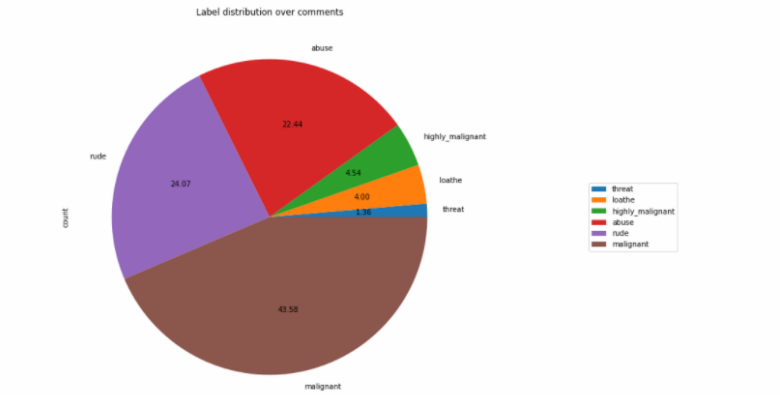


Malignant comments are the highest among all whereas threat comments are very less. Rude and abuse comments are also present more.

Plotting pie-chart

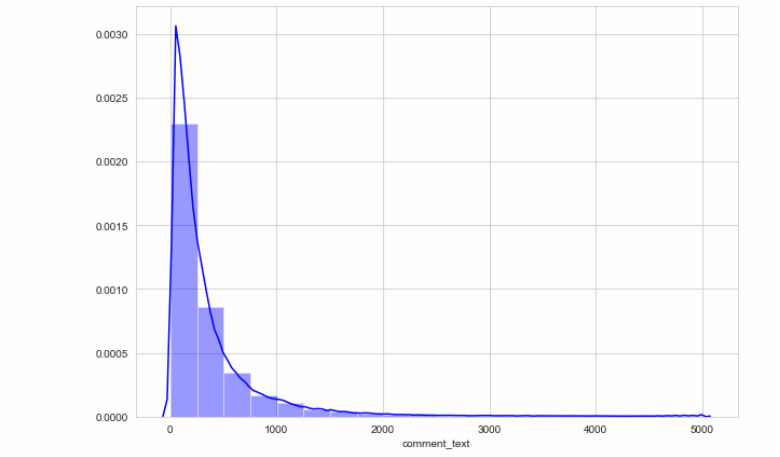
```
In [24]: # Visualizing the label distribution of comments using pie chart
comments_labels = ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']
df_distribution = df_train[comments_labels].sum()\
                            .to_frame()\
                            .rename(columns={0: '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))
```

```
Out[24]: <matplotlib.legend.Legend at 0x2500b62ad88>
```




```
In [25]: # 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 = 'Blue')
```

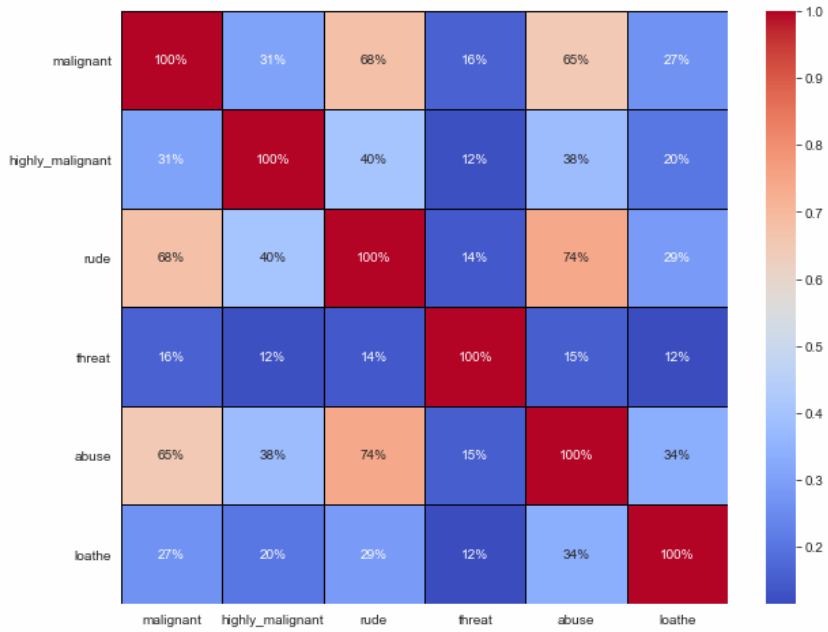
```
Out[25]: <AxesSubplot:xlabel='comment_text'>
```



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 [29]: # Plotting heatmap for visualizing the correlation
         plt.figure(figsize=(10,8))
         sns.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',cmap='coolwarm',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.

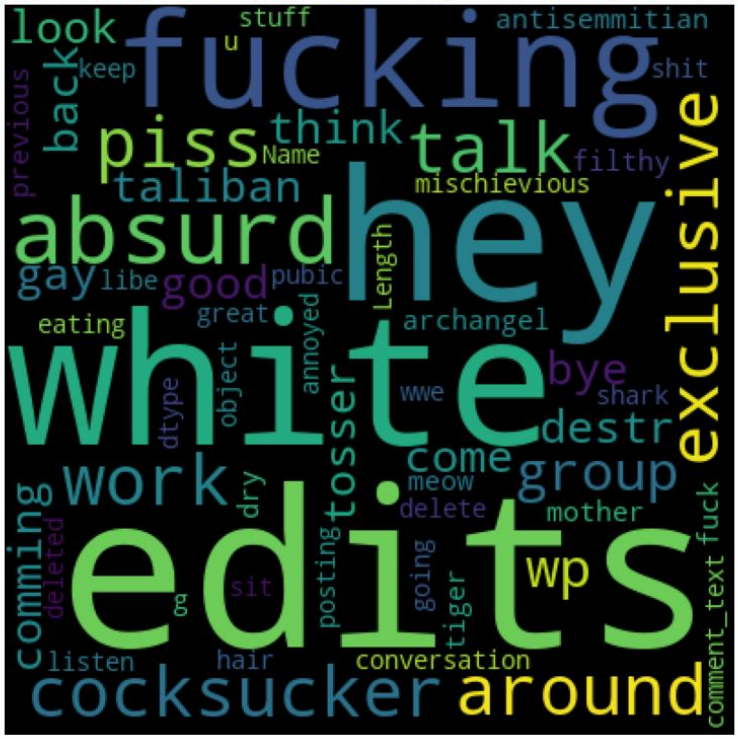
Cleaning the data using NLP

- 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

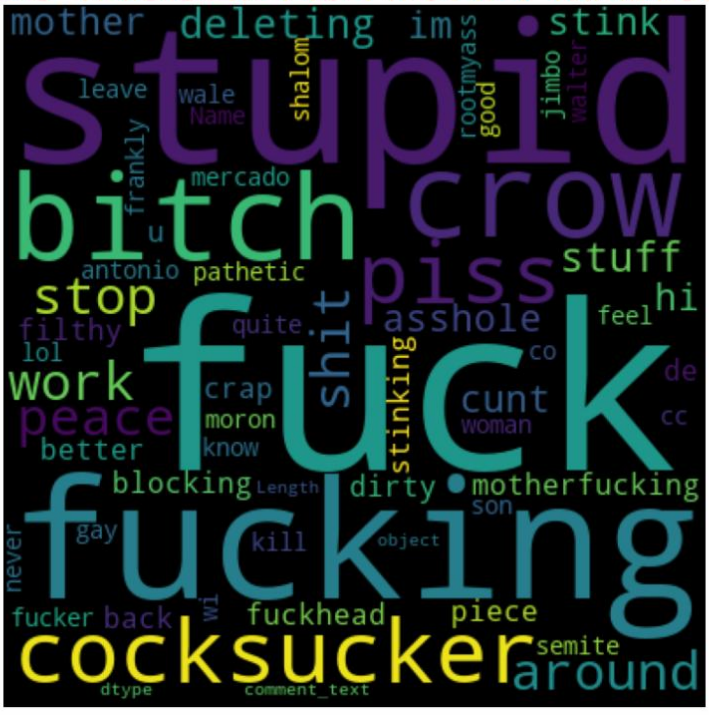
In [45]:
```
# Function Definition for using regex operations and other text preprocessing for getting cleaned texts
def clean_comments(text):

    #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)
```

➢ **Plotting wordcloud for each feature**

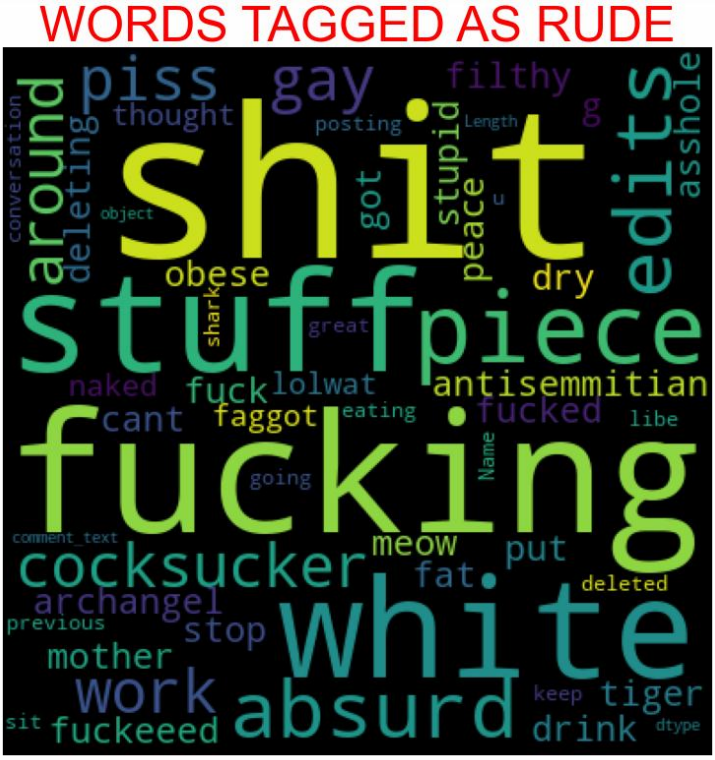
WORDS TAGGED AS MALIGNANT



WORDS TAGGED AS HIGHLY MALIGNANT

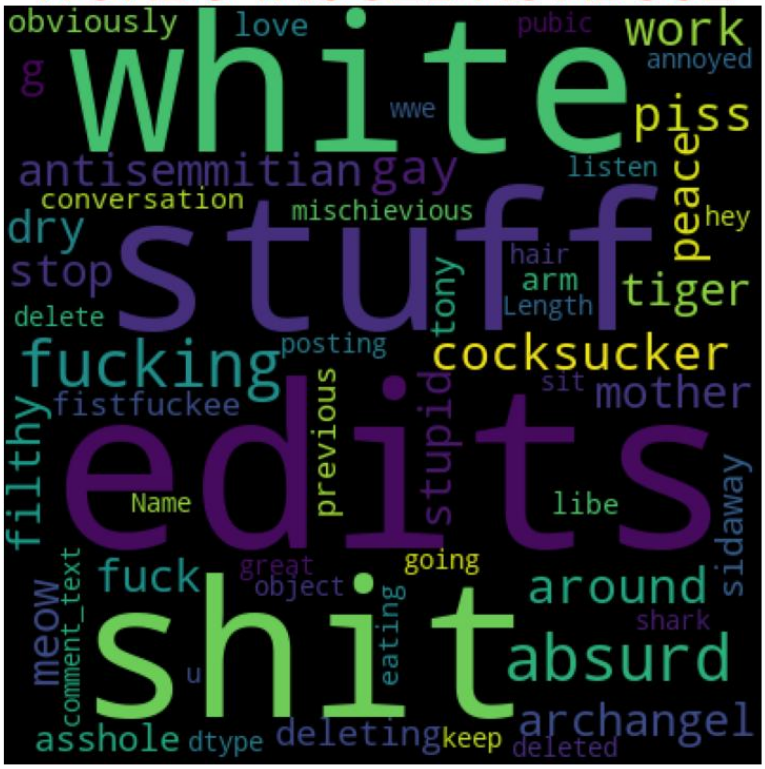


WORDS TAGGED AS RUDE



WORDS TAGGED AS THREAT

WORDS TAGGED AS ABUSE



WORDS TAGGED AS LOATHE

MODEL/S DEVELOPMENT AND EVALUATION

Separating independent and dependent variables

In [64]:
```
# Converting the features into number vectors
tf_vec = TfidfVectorizer(max_features = 15000, stop_words='english')
```

In [65]:
```
# 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'])
```

In [66]:
```
y=df_train['label']
```

In [67]:
```
# Checking the shape of the data
print(X.shape,'\t\t', y.shape)
```

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

In [68]:
```
# Doing the above process for test data
test_vec = tf_vec.fit_transform(df_test['comment_text'])
test_vec
```

Out[68]:
```
<153164x15000 sparse matrix of type '<class 'numpy.float64'>'
        with 2870432 stored elements in Compressed Sparse Row format>
```

In [69]:
```
test_vec.shape
```

Out[69]:
```
(153164, 15000)
```

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

In [70]:
```
# 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 [71]:
```
# Checking the shape of x data
print(x_train.shape,'\t\t',x_test.shape)
```

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

In [72]:
```
# Checking the shape of y data
print(y_train.shape,'\t',y_test.shape)
```

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

Handling the imbalanced data using oversampling technique

In [73]:
```
# Importing the Oversampling library and Counter
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
```

In [74]:
```
# 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})
```

Splitting the data:

Using scikit learn library of python we have imported train test split to divide data into train data and test data. We have use test size is 0.3 (30%), rest of 0.7 (70%) data as train data. Random state provides us a seed to the random number generator by using following codes.

```
from sklearn.model_selection import train_train_split
x_train, x_test, y_train, y_test = train_train_split (x, y, test_size=0.3, random_state=42)
```

```
In [70]: # 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)
```

Building Machine Learning Algorithms:

The following classifier algorithms we used are:

- **Logistic Regression**

Logistic Regression is a supervised machine algorithm, which is used to solve classification problems.

The target variable of Logistic Regression is discrete (binary or ordinal). Predicted values Logistic Regression are the probability of the particular levels of the given values of the input variable.

- **GaussianNB**

Naive Bayes are the group of supervised machine classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality. They find use when the dimensionality of the inputs is high. Complex classification problems can also be implemented by using GaussianNB Classifier, generall known as Naïve Bayes Classifier.

- **Decision Tree Classifier**

Decision Tree is a supervised machine algorithm, which is used to solve for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- **KNeighbors Classifier**

K-NN algorithm is a supervised machine learning algorithm, which is used to solve for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

K-NN algorithm is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also known as a lazy learner algorithm because it does not learn from the training set immediately instead it

stores the data and at the time of classification, it performs an action on the dataset.

```
In [77]: # Initializing the instance of the model
         LR=LogisticRegression()
         mnb=MultinomialNB()
         dtc=DecisionTreeClassifier()
         knc=KNeighborsClassifier()
         rfc=RandomForestClassifier()
         abc=AdaBoostClassifier()
         gbc=GradientBoostingClassifier()
```

```
In [78]: models= []
         models.append(('Logistic Regression',LR))
         models.append(('MultinomialNB',mnb))
         models.append(('DecisionTreeClassifier',dtc))
         models.append(('KNeighborsClassifier',knc))
         models.append(('RandomForestClassifier',rfc))
         models.append(('AdaBoostClassifier',abc))
         models.append(('GradientBoostingClassifier',gbc))
```

Evaluation of ML Models:

- As you can see , I had called the algorithms, then I called the empty list with the name models [ ], and calling all the model one by one and storing the result in that.

- We can observe that I imported the metrics to find the accuracy score, roc_auc_curve, auc, confusion_matrix and classification_report in order to interpret the model's output. Then I also selected the model to find the cross_validation_score value.

```
In [68]: models = []
         models.append(('LogisticRegression',LR))
         models.append(('GaussianNB',gnb))
         models.append(('DecisionTreeClassifier',dtc))
         models.append(('KNeighborsClassifier',knc))
```

```
In [69]: #Importing required modules
         from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report
         from sklearn.model_selection import cross_val_score
```

Key Metrics for success in solving problem under consideration:

Accuracy score

Accuracy is one of the matrix foe evaluating classification models. Informally, accuracy is the fraction of predictions our model got right.

Formally, accuracy has the following definition.

$$Accuracy = \frac{\text{No. of correct predictions}}{\text{Total No. of predictions}} \times 100$$

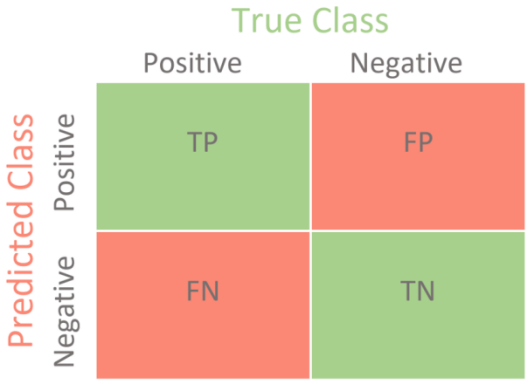
Accuracy score express in percentage.

Confusion matrix

Confusion matrix is a matrix used to determine the performance of classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood but the related terminologies may be confusing, since it shows the errors in the model performance in the form of confusion matrix.

The matrix is divided into two dimensions that are Predicted values and Actual values along with the total number of predictions.

Predicted values are those values, which are predicted by the model, and Actual values are the true value for the given observations.



The above matrix is said to be confusion matrix, has the following cases.

- True Positive (TP): The model has predicted True (Yes or 1) and the actual value was also True (Yes or 1).

- False Positive (FP): The model has predicted True (Yes or 1) but the actual value was False (No or 0). It is called a Type-I error.

- False Negative (FN): The model has predicted False (No or 0) but the actual value was True (Yes or 1). It is called a Type-II error.

- ❖ True Negative (TN): The model has predicted False (No or 0) and the actual value was also False (No or 0).

Classification report

A classification report is a performance evaluation metric in machine learning. It is used to show model's precision, recall. F1 Score and support.

Precision: Precision is defined as the ratio of true positive (TP) to the sum of true and false positives (TP+FP).

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

Recall: Recall is defined as the ratio of true positive (TP) to the sum of true positive (TP) and false negative (FN) . It is also known as Sensitivity.

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

F1 Score: The F1 is the harmonic mean of precision and recall. The closer the value of F1 Score is to 1.0 is the better expected performance of the model.

$$F1\ Score = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Support: Support is the number of actual occurrences of the class in the dataset. It just diagnoses the performance evaluation process.

Accuracy: The sum of true positives and true negatives (TP + TN) divided by the total number of samples. This is only accurate if the model is balanced. It will give inaccurate results if there is a class imbalance.

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

Macro average: A macro average will compute the metric independently for each class and then take average hence treating all classes equally.

Weighted average : The weighted average F1 score is calculated by take the mean of all per-class F1 scores while considering each class's support.

Cross Validation

Cross validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how statistical model generalizes to an independent dataset.

AUC – ROC Curve:

AUC – ROC (Area Under the Curve – Receiver Operating Characteristic) curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

The ROC curve is plotted with TPR (True Positive Rate) against the FPR (False Positive Rate) where TPR is on the y-axis and FPR on the x-axis.

$$TPR\ (or)\ Recall\ (or)\ Sensitivity = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$Specificity = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

$$FPR = 1 - Specificity = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}$$

Evaluation of ML Models

```
In [81]: # 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')
```

As you can observe, I made a for loop and called all the algorithms one by one and appending their result to models. The same I had done to store roc_auc_curve, auc score, and cross validation score. Let me show the output so that we can glance the result in more appropriate way.

Result of ML Models:

We saved result of ML Models in DataFrame.

```
In [82]: # 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[82]:

	Model	Accuracy_score	Cross_val_score	roc_auc_score	Hamming_loss	Log_loss
0	Logistic Regression	94.453961	95.609478	89.608080	0.055460	1.915565
1	MultinomialNB	91.026070	94.660057	88.629183	0.089739	3.099546
2	DecisionTreeClassifier	92.835060	94.091031	83.770207	0.071649	2.474717
3	KNeighborsClassifier	77.466995	91.786101	66.472215	0.225330	7.782766
4	RandomForestClassifier	95.318767	95.697839	83.139663	0.046812	1.616852
5	AdaBoostClassifier	92.697193	94.591749	81.607629	0.073028	2.522332
6	GradientBoostingClassifier	94.422627	94.034630	79.543991	0.055774	1.926369

From the above algorithms, we can say that the RandomForestClassifier is working well by giving an accuracy of **95.31%** and cross validation score of **95.69%**. Now we will perform Hyperparameter Tuning to improve accuracy of our model (i.e, RandomForestClassifier)

Hyperparameter Tuning:

In order to increase the accuracy score of the model we use hyperparameter tuning of the best model in order to find best parameters by using GridSearchCV() .

Random Forest Classifier

In [83]:
```
# Creating parameter list to pass in GridSearchCV
parameters={'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10],'n_estimators': [50, 100, 500]}
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV
rfc=RandomForestClassifier(random_state=42)
rfc=GridSearchCV(rfc,parameters,cv=3,scoring='accuracy')
rfc.fit(x_train_os,y_train_os)
print(rfc.best_params_)
print(rfc.best_score_)
```

NOTE:

If we run GridSearchCV and RandomSearchCV, it takes more than 6 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.

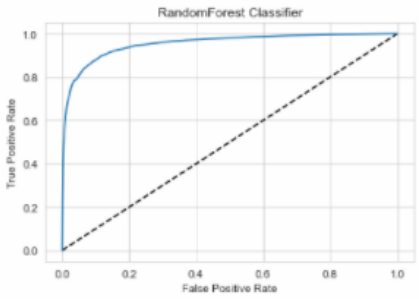
In [85]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train_os,y_train_os)
```

Out[85]:
```
RandomForestClassifier()
```

In [86]:
```
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:  94.4226270053476
Cross validation score:  95.70598627552735
roc_auc_score:  0.7954399074837164
Hamming_loss: 0.05577372994652406
Log loss: 1.9263691649909267
Classification report:

              precision    recall  f1-score   support

           0       0.96      0.98      0.97     43004
           1       0.79      0.61      0.69      4868

    accuracy                           0.94     47872
   macro avg       0.88      0.80      0.83     47872
weighted avg       0.94      0.94      0.94     47872

Confusion matrix:

[[42239   765]
 [ 1905  2963]]
```

In [87]:
```
# 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)
```




```
0.8312202541476909
```

Saving the model

In order to dump the model which we have developed so that we can use it to make predictions in future, we have saved or dumped the best model.

Conclusion:

- 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 was 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 7 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 7 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.