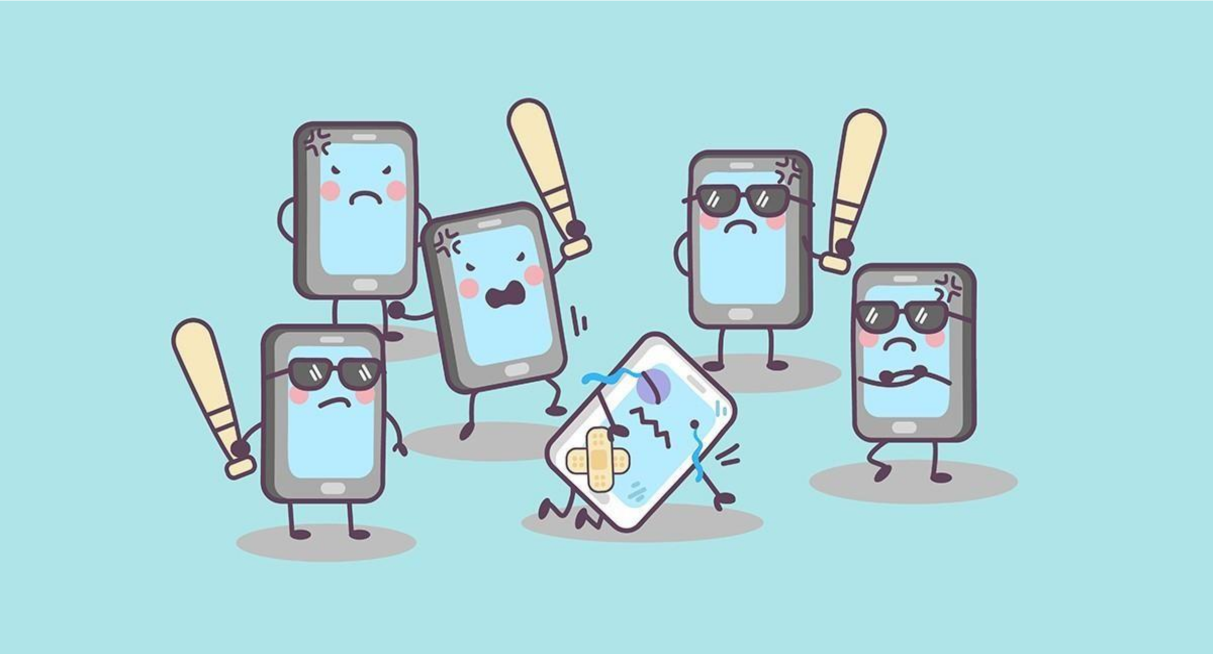
FLIP ROBO

MALIGNANT COMMENTS CLASSIFICATION



Prepared By:
Abhishek Ranjan

SME Name:
Mohd. Kashif

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project using NLP "***MALIGNANT COMMENTS CLASSIFICATION***" and also want to thanks my SME for providing the dataset and helping me to complete this project. This project would not have been accomplished without their help and insights.

I would also like to thank my academic "Data Trained Education" and their team who has helped me to learn Machine Learning and NLP.

INTRODUCTION

1. Business Problem Framing

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 cyber bullying.

2. Conceptual Background of the Domain Problem

There has been a remarkable increase in the cases of cyber bullying 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.

3. Review of Literature

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, cyber bullying, 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 behavior.

4. Motivation for the Problem Undertaken

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 inoffensive, but “u are an idiot” is clearly offensive.

Analytical Problem Framing

1. Mathematical/ Analytical Modeling of the Problem

1) Used Panda’s Library to save data into csv file
2) Cleaned Data by removing irrelevant features
3) Descriptive Statistics
4) Analyzed correlation
5) Converted all messages to lower case
6) Replaced email addresses with 'email'
7) Replaced URLs with 'webaddress'
8) Replaced money symbols with 'moneysymb'
   (£ can by typed with ALT key + 156)
9) Replaced 10digit phone numbers (formats include parenthesis, spaces, no spaces, dashes) with 'phone number'
10) Replace Numbers with 'number'
11) Removed Punctuation
12) Replaced extra space
13) Replaced leading and trailing white space
14) Removed \n
15) Added and removed stop words
16) Words of Sentence
17) Calculated length of sentence
18) Made one Target Column
19) Removed Total length
20) Checked the word which are offensive using WordCloud
21) Checked the word which are not offensive using WordCloud
22) Converted text into vectors using TF-IDF

2. Data Sources and their formats

There are two data-set in csv format: **train and test dataset**. Features of this dataset are:

- 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.

3. Data Pre-processing:

a) Checked Top 5 Rows of both Dataset

```
comments_train.head()
```

	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

```
comments_test.head()
```

	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.

b) Checked Total Numbers of Rows and Column

```
comments_train.shape
```

(159571, 8)

```
comments_test.shape
```

(153164, 2)

c) Checked All Column Name

```
comments_train.columns
```

```
Index(['id', 'comment_text', 'malignant', 'highly_malignant', 'rude', 'threat',
       'abuse', 'loathe'],
      dtype='object')
```

```
comments_test.columns
```

```
Index(['id', 'comment_text'], dtype='object')
```

d) Checked Data Type of All Data

```
comments_train.dtypes
```

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

```
comments_test.dtypes
```

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

e) Checked for Null Values

```
comments_train.isnull().sum()
```

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

```
comments_test.isnull().sum()
```

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

There is no null value in the dataset.

f) Checking if "-" values present in dataset or not

```
(comments_train=='?').sum()
```

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

```
(comments_test=='?').sum()
```

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

g) Checked total number of unique values

```
comments_train.nunique()
```

```
id                  159571
comment_text        159571
malignant                2
highly_malignant         2
rude                     2
threat                   2
abuse                    2
loathe                   2
dtype: int64
```

```
comments_test.nunique()
```

```
id              153164
comment_text    153164
dtype: int64
```

h) Checking unique values present in the columns: ("malignant", "highly_malignant", "rude", "threat", "abuse", "loathe")

```
comment_columns= ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
for i in comments_train[comment_columns]:
    print(i, comments_train[i].unique(),"\n")
```

```
malignant [0 1]

highly_malignant [0 1]

rude [0 1]

threat [0 1]

abuse [0 1]

loathe [0 1]
```

i) Information about Data

```
comments_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   id                159571 non-null  object
 1   comment_text      159571 non-null  object
 2   malignant         159571 non-null  int64 
 3   highly_malignant  159571 non-null  int64 
 4   rude              159571 non-null  int64 
 5   threat            159571 non-null  int64 
 6   abuse             159571 non-null  int64 
 7   loathe            159571 non-null  int64 
dtypes: int64(6), object(2)
memory usage: 9.7+ MB
```

```
comments_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153164 entries, 0 to 153163
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            153164 non-null  object
 1   comment_text  153164 non-null  object
dtypes: object(2)
memory usage: 2.3+ MB
```

j) Data cleaning

- Dropped Column " id" as this column contains serial no.

```
#dropping column "id" as this column contains unique value which is not relevant for prediction
comments_train.drop("id",axis=1,inplace=True)
```

```
#dropping column "id" as this column contains unique value which is not relevant for prediction
comments_test.drop("id",axis=1,inplace=True)
```

k) Data Visualization

i. Uni-Variate Analysis
   - ➢ Used Countplot

ii. Bivariate Analysis
   (For comparison between each feature with target)
   - ➢ Used Bar plot

iii. Multivariate Analysis
   (For comparison between all features with target)
   - ➢ Used Pair plot

4. Data Inputs- Logic- Output Relationships

I. Descriptive Statistics

```
# Description of comments_train Dataset : works only on continuous column
comments_train.describe()
```

	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

Checking Description through heatmap

```
plt.figure(figsize=(10,8))
sns.heatmap(round(comments_train.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt='f')
plt.xticks(fontsize=18)
plt.xticks(fontsize=12)
plt.title('variables')
plt.show()
```

Observation:

```
- We can see total count is 159571.000000 of each columns which shows there is no null values.
- Total rows are 159571
- Total continuous columns are: 6 and categorical column are: 2
- Std deviation is more than mean
- All percentile difference is 0
```

II. Checking Correlation

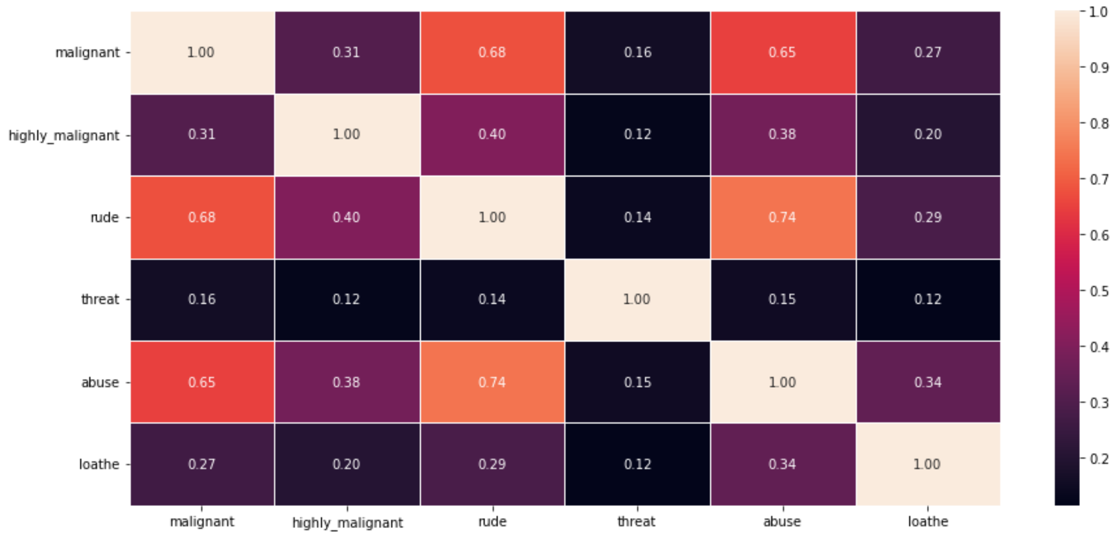
```
comments_train.corr()
```

	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

This gives the correlation between the dependent and independent variables.

Checking correlation with heatmap

```
plt.figure(figsize=(15,7))
sns.heatmap(comments_train.corr(),annot=True, linewidth=0.5, linecolor='white', fmt='.2f')
```



Outcome of Correlation:

- malignant and highly_malignant has 31 percent correlation with each other and positively correlated.
- highly_malignant and rude has 40 percent correlation with each other and positively correlated.
- rude and threat has 14 percent correlation with each other and positively correlated.
- threat and abuse has 15 percent correlation with each other and positively correlated.
- abuse and loathe has 34 percent correlation with each other and positively correlated.

- Max correlation is between highly_malignant and rude
- Min correlation is between rude and threat

III. Handling "cmnt_train" Dataset

```
cmnt_train['length'] = cmnt_train['comment_text'].str.len()
```

```
# Convert all messages to lower case
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.lower()
```

```
# Replace email addresses with 'email'
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                                                    'emailaddress')
```

```
# Replace URLs with 'webaddress'
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                                                    'webaddress')
```

```
# Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.replace(r'£|\$', 'dollers')
```

```
# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                                                    'phonenumber')
```

```
# Replace numbers with 'numbr'
cmnt_train['comment_text'] = cmnt_train['comment_text'].str.replace(r'\d+(\.\d+)?',
                                                                  'numbr')
```

```
#remove  punctuation
cmnt_train["comment_text"]=cmnt_train["comment_text"].str.replace(r'[^\w\d\s]'," ")
```

```
# replace extra space
cmnt_train["comment_text"]=cmnt_train["comment_text"].str.replace(r'^\s+'," ")
```

```
#replacing leadning and trailing white space
cmnt_train["comment_text"]=cmnt_train["comment_text"].str.replace(r'^\s+|\s+?$', "")
```

```
#removing \n
cmnt_train["comment_text"]=cmnt_train["comment_text"].str.replace("\n"," ")
```

```
cmnt_train['comment_text'] = cmnt_train['comment_text'].apply(lambda x: ' '.join
                                                              (term for term in x.split() if term not in string.punctuation))
```

```
stop_words = set(stopwords.words('english') + ['m','ur','aww','d','cant','doin','ja','u','ü','ur','4','2','im',
                                               'dont','doin','ure'])
```

```
cmnt_train['comment_text'] = cmnt_train['comment_text'].apply(lambda x: ' '.join
                                                              (term for term in x.split() if term not in stop_words))
```

```
#Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item.

lem=WordNetLemmatizer()
cmnt_train['comment_text'] = cmnt_train['comment_text'].apply(lambda x: ' '.join
                                                              (lem.lemmatize(t) for t in x.split()))
```

```
cmnt_train['clean_length'] = cmnt_train.comment_text.str.len()
cmnt_train.head()
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length	clean_length
0	explanation edits made username hardcore metal...	0	0	0	0	0	0	264	168
1	match background colour seemingly stuck thanks...	0	0	0	0	0	0	112	87
2	hey man really trying edit war guy constantly ...	0	0	0	0	0	0	233	141
3	make real suggestion improvement wondered sect...	0	0	0	0	0	0	622	365
4	sir hero chance remember page	0	0	0	0	0	0	67	29

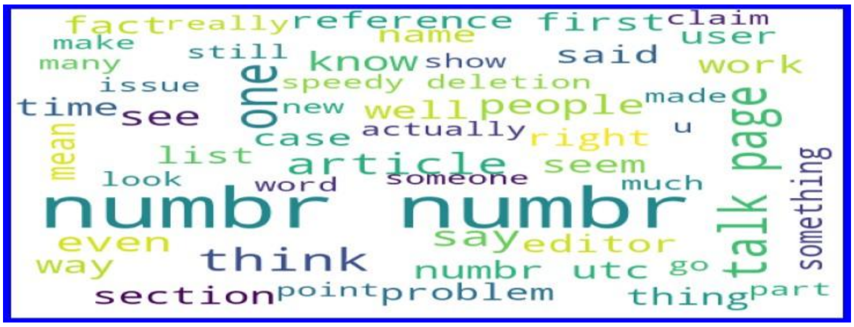
```python
# making one target column
cmnt_train["label"]= cmnt_train["malignant"] | cmnt_train["highly_malignant"] | cmnt_train["rude"] |
                     cmnt_train["threat"] | cmnt_train["abuse"] | cmnt_train["loathe"]
```

```python
# Total length removal
print ('Origian Length', cmnt_train.length.sum())
print ('Clean Length', cmnt_train.clean_length.sum())
```

```
Origian Length 62893130
Clean Length 39961030
```

```python
# seeing the word which are offensive
from wordcloud import WordCloud

malignant=cmnt_train["comment_text"][cmnt_train["label"]==1]

malign_cloud= WordCloud(width=600,height=400,background_color="black",max_words=50).generate(" ".join(malignant))

plt.figure(figsize=(10,8),facecolor="k")
plt.imshow(malign_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```


```python
# seeing the word which are not offensive
from wordcloud import WordCloud

non_malignant=cmnt_train["comment_text"][cmnt_train["label"]==0]

malign_cloud= WordCloud(width=600,height=400,background_color="white",max_words=50).generate(" ".join(non_malignant))

plt.figure(figsize=(10,8),facecolor="b")
plt.imshow(malign_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```




```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
#Convert text into vectors using TF-IDF
tf_vec = TfidfVectorizer(max_features = 10000, stop_words='english')
features = tf_vec.fit_transform(cmnt_train['comment_text'])
x = features
y=cmnt_train['label']
```

IV. Handling "cmnt_test" Dataset

```
cmnt_test= comments_test.copy()
```

```
cmnt_test["comment_text"] = cmnt_test["comment_text"].str.lower()
```

```
cmnt_test["length"] = cmnt_test["comment_text"].str.len()
```

```
#replacing with email address
cmnt_test["comment_text"]= cmnt_test["comment_text"].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',"emailaddress")
```

```
#replacing with web address
cmnt_test["comment_text"]= cmnt_test["comment_text"].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\s*)?$',
                                                              "webaddress")
```

```
#replacing with number
cmnt_test["comment_text"]= cmnt_test["comment_text"].str.replace(r'\d+(\.\d+)?',"number")
```

```
#remove  punctuation
cmnt_test["comment_text"]=cmnt_test["comment_text"].str.replace(r'[^\w\d\s]'," ")
```

```
# replace extra space
cmnt_test["comment_text"]=cmnt_test["comment_text"].str.replace(r'^\s+'," ")
```

```
#replacing leading and trailing white space
cmnt_test["comment_text"]=cmnt_test["comment_text"].str.replace(r'^\s+|\s+?$', "")
```

```
#replacing with\n
cmnt_test["comment_text"]=cmnt_test["comment_text"].str.replace("\n"," ")
```

```
# remove stopwords
stop_words = set(stopwords.words('english') + ["m","ur","aww","d","dont","cant","doin","ja","u"])
```

```
cmnt_test["comment_text"]= cmnt_test["comment_text"].apply(lambda x: ' '.join
                                                          (term for term in x.split() if term not in stop_words ))
```

```
#Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item.

lem=WordNetLemmatizer()
cmnt_test['comment_text'] = cmnt_test['comment_text'].apply(lambda x: ' '.join
                                                           (lem.lemmatize(word) for word in x.split()))
```

```
cmnt_test["clean_cmnt_test"] = cmnt_test["comment_text"].str.len()
cmnt_test.head()
```

	comment_text	length	clean_cmnt_test
0	yo bitch rule succesful ever whats hating sad ...	367	221
1	rfc title fine imo	50	18
2	source zawe ashton lapland	54	26
3	look back source information updated correct f...	205	109
4	anonymously edit article	41	24

```
print('original length',cmnt_test.length.sum())
print('cleaned length',cmnt_test.clean_cmnt_test.sum())
```

```
original length 55886104
cleaned length 35617170
```

```
cmnt_train.shape
```

```
(159571, 10)
```

```
cmnt_test.shape
```

```
(153164, 3)
```

5. State the set of assumptions (if any) related to the problem under consideration

- It was observed that there is one column “id” which is irrelevant column as it contains serial no, so, have to drop this column.
- It was observed that in columns there are irrelevant values present in comment_text. So, we need to drop, replace and remove those values.
- Also have to convert comment_text into vectors using TF-IDF
- Have to create on Target column also.

6. Hardware and Software Requirements and Tools Used

- **Hardware used:**
  - **Processor**: 11th Gen Intel(R) Core (TM) i3-1125G4 @ 2.00GHz 2.00 GHz
  - **System Type**: 64-bit OS
- **Software used:**
  - **Anaconda** for 64-bit OS
  - **Jupyter** notebook

- **Tools, Libraries and Packages used:**

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy.stats import zscore
from sklearn.preprocessing  import power_transform, StandardScaler, LabelEncoder
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score,classification_report, confusion_matrix, plot_roc_curve
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import BaggingClassifier,AdaBoostClassifier

import nltk
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import gensim
from gensim.models import Word2Vec
from sklearn.feature_extraction.text import TfidfVectorizer

import pickle
import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Model/s Development and Evaluation

1. Identification of possible problem-solving approaches (methods)

In this project, we want to differentiate between comments and its categories and for this we have used these approaches:

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Checked for special character present in dataset or not
- Checked total number of unique values
- Information about Data
- Checked Description of Data and Dataset
- Dropped irrelevant Columns
- Replaced special characters and irrelevant data
- Checked all features through visualization.
- Checked correlation of features
- Converted all messages to lower case
- Replaced email addresses with 'email'
- Replaced URLs with 'webaddress'
- Replaced money symbols with 'moneysymb'
  (£ can by typed with ALT key + 156)
- Replaced 10digit phone numbers (formats include parenthesis, spaces, no spaces, dashes) with 'phone number'
- Replace Numbers with 'number'
- Removed Punctuation
- Replaced extra space
- Replaced leading and trailing white space
- Removed \n
- Added and removed stop words
- Words of Sentence
- Calculated length of sentence
- Made one Target Column

- Removed Total length
- Checked the word which are offensive using WordCloud
- Checked the word which are not offensive using WordCloud
- Converted text into vectors using TF-IDF

Testing of Identified Approaches (Algorithms)

1. Logistic Regression
2. AdaBoost Classifier
3. Decision Tree Classifier
4. KNN Classifier
5. Gradient Boosting Classifier
6. XGB Classifier
7. MultinomialNB

2. Run and evaluate selected models

Creating Model

We are using Classification Algorithm

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=70)
```

```
x.shape, y.shape
```

```
((159571, 10000), (159571,))
```

```
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

```
((111699, 10000), (111699,), (47872, 10000), (47872,))
```

We can see the x.shape value is divided into x_train.shape and x_test.shape and like this y.shape is also divided. We will understand this by Classification problem.

1. Logistic Regression

```
lr=LogisticRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)

print("accuracy_score: ", accuracy_score(y_test, pred_lr))
print("confusion_matrix: \n", confusion_matrix(y_test, pred_lr))
print("classification_report: \n", classification_report(y_test,pred_lr))
```

```
accuracy_score:  0.956488135026738
confusion_matrix:
 [[42774   253]
 [ 1830  3015]]
classification_report:
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.74      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
```

2. AdaBoost Classifier

```
abc = AdaBoostClassifier()
abc.fit(x_train,y_train)
pred_abc = abc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_abc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_abc))
print("classification_report: \n",classification_report(y_test,pred_abc))
```

```
accuracy_score:  0.9472760695187166
confusion_matrix:
 [[42637   390]
 [ 2134  2711]]
classification_report:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97     43027
           1       0.87      0.56      0.68      4845

    accuracy                           0.95     47872
   macro avg       0.91      0.78      0.83     47872
weighted avg       0.94      0.95      0.94     47872
```

3. Decision Tree Classifier

```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc = dtc.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_dtc))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_dtc))
print("classification_report: \n",classification_report(y_test,pred_dtc))
```

```
accuracy_score:  0.940863135026738
confusion_matrix:
 [[41653  1374]
 [ 1457  3388]]
classification_report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97     43027
           1       0.71      0.70      0.71      4845

    accuracy                           0.94     47872
   macro avg       0.84      0.83      0.84     47872
weighted avg       0.94      0.94      0.94     47872
```

4. KNN Classifier

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_knn))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_knn))
print("classification_report: \n",classification_report(y_test,pred_knn))
```

```
accuracy_score:  0.914480280748663
confusion_matrix: 
 [[42404   623]
 [ 3471  1374]]
classification_report: 
               precision    recall  f1-score   support

           0       0.92      0.99      0.95     43027
           1       0.69      0.28      0.40      4845

    accuracy                           0.91     47872
   macro avg       0.81      0.63      0.68     47872
weighted avg       0.90      0.91      0.90     47872
```

5. Gradient Boosting Classifier

```
gb = GradientBoostingClassifier(n_estimators =100,learning_rate=0.1, max_depth=4)
gb.fit(x_train,y_train)
pred_gb = gb.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_gb))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_gb))
print("classification_report: \n",classification_report(y_test,pred_gb))
```

```
accuracy_score:  0.943829378342246
confusion_matrix: 
 [[42869   158]
 [ 2531  2314]]
classification_report: 
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     43027
           1       0.94      0.48      0.63      4845

    accuracy                           0.94     47872
   macro avg       0.94      0.74      0.80     47872
weighted avg       0.94      0.94      0.94     47872
```

6. XGB Classifier

```
XGBC= XGBClassifier()
XGBC.fit(x_train,y_train)
pred_XGBC = XGBC.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_XGBC))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_XGBC))
print("classification_report: \n",classification_report(y_test,pred_XGBC))
```

```
accuracy_score:  0.9540858957219251
confusion_matrix: 
 [[42737   290]
 [ 1908  2937]]
classification_report: 
               precision    recall  f1-score   support

           0       0.96      0.99      0.97     43027
           1       0.91      0.61      0.73      4845

    accuracy                           0.95     47872
   macro avg       0.93      0.80      0.85     47872
weighted avg       0.95      0.95      0.95     47872
```

7. MultinomialNB

```
MNB= MultinomialNB()
MNB.fit(x_train,y_train)
pred_MNB = MNB.predict(x_test)

print("accuracy_score: ",accuracy_score(y_test, pred_MNB))
print("confusion_matrix: \n",confusion_matrix(y_test, pred_MNB))
print("classification_report: \n",classification_report(y_test,pred_MNB))
```

```
accuracy_score:  0.9484040775401069
confusion_matrix:
 [[42860   167]
 [ 2303  2542]]
classification_report:
               precision    recall  f1-score   support

           0       0.95      1.00      0.97     43027
           1       0.94      0.52      0.67      4845

    accuracy                           0.95     47872
   macro avg       0.94      0.76      0.82     47872
weighted avg       0.95      0.95      0.94     47872
```

Cross Validation Score for all the model

```
#CV Score for Logistic Regression
print('CV score for Logistic Regression: ',cross_val_score(lr,x,y,cv=5).mean())
```

```
CV score for Logistic Regression:  0.9561637086494331
```

```
#CV Score for AdaBoost Classifier
print('CV score for AdaBoost Classifier: ',cross_val_score(abc,x,y,cv=5).mean())
```

```
CV score for AdaBoost Classifier:  0.946199496019436
```

```
#CV Score for Decision Tree Classifier
print('CV score for Decision Tree Classifier: ',cross_val_score(dtc,x,y,cv=5).mean())
```

```
CV score for Decision Tree Classifier:  0.940665908185353
```

```
#CV Score for KNN Classifier
print('CV score for KNN Classifier: ',cross_val_score(knn,x,y,cv=5).mean())
```

```
CV score for KNN Classifier:  0.9165637798065145
```

```
#CV Score for Gradient Boosting Classifier
print('CV score for Gradient Boosting Classifier: ',cross_val_score(gb,x,y,cv=5).mean())
```

```
CV score for Gradient Boosting Classifier:  0.9436363725082835
```

```
#CV Score for XGB Classifier
print('CV score for XGB Classifier: ',cross_val_score(XGBC,x,y,cv=5).mean())
```

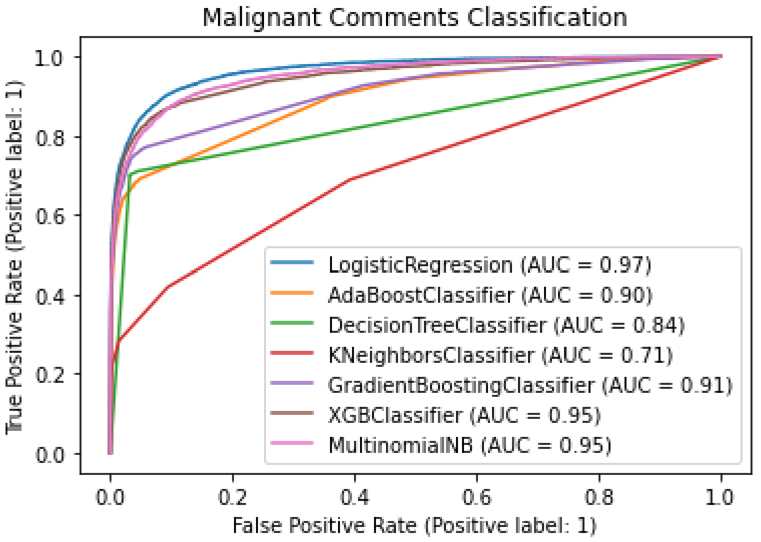
```
CV score for XGB Classifier:  0.953794860582693
```

```
#CV Score for MultinomialNB Classifier
print('CV score for MultinomialNB: ',cross_val_score(MNB,x,y,cv=5).mean())
```

```
CV score for MultinomialNB:  0.9476909979684454
```

ROC & AUC Curve for all model

```
#Lets plot roc curve and check auc and performance of all algorithms
disp = plot_roc_curve(lr, x_test, y_test)
plot_roc_curve(abc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(dtc, x_test, y_test, ax = disp.ax_)
plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
plot_roc_curve(gb, x_test, y_test, ax = disp.ax_)
plot_roc_curve(XGBC, x_test, y_test, ax = disp.ax_)
plot_roc_curve(MNB, x_test, y_test, ax = disp.ax_)
plt.title("Malignant Comments Classification")
plt.legend(prop={"size" :10} ,loc = 'lower right')
plt.show()
```



From the observation of accuracy and cross validation score and their difference we can predict that Logistic Regression is the best model.

Hyper parameter tuning for best model

The Logistic Regression with GridsearchCV

```
solver_options = ['newton-cg', 'lbfgs', 'liblinear', 'sag']
multi_class_options = ['ovr', 'multinomial']
class_weight_options = ['None', 'balanced']
```

```
param_grid = dict(solver = solver_options,
                  multi_class = multi_class_options,
                  class_weight = class_weight_options)
```

```
clf = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring = 'accuracy', )
```

```
clf.fit(x,y)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'class_weight': ['None', 'balanced'],
                         'multi_class': ['ovr', 'multinomial'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag']},
             scoring='accuracy')
```

```
clf.best_estimator_
```

```
LogisticRegression(class_weight='None', multi_class='ovr', solver='newton-cg')
```

```
print (f'Accuracy - : {clf.score(x,y)}')
```

```
Accuracy - : 0.9605128751464865
```

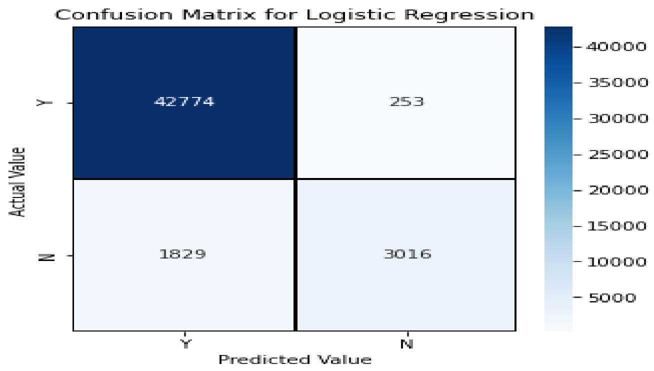
```
malignant= LogisticRegression(class_weight='None',multi_class='ovr')
malignant.fit(x_train,y_train)
```

```
LogisticRegression(class_weight='None', multi_class='ovr')
```

```
pred = malignant.predict(x_test)
print("accuracy score: ",accuracy_score(y_test,pred))
print("Cross_validation_Score :", cross_val_score(lr,x,y,cv=5).mean())
print("confusion_matrix: \n",confusion_matrix(y_test,pred))
print("classification_report: \n",classification_report(y_test,pred))
```

```
accuracy score:  0.9565090240641712
Cross_validation_Score : 0.9561637086494331
confusion_matrix: 
 [[42774   253]
 [ 1829  3016]]
classification_report: 
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.74      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
```

```
cm = confusion_matrix(y_test, pred)

x_axis_labels = ["Y","N"]
y_axis_labels = ["Y","N"]

f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Blues",
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("Predicted Value")
plt.ylabel("Actual Value ")
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```

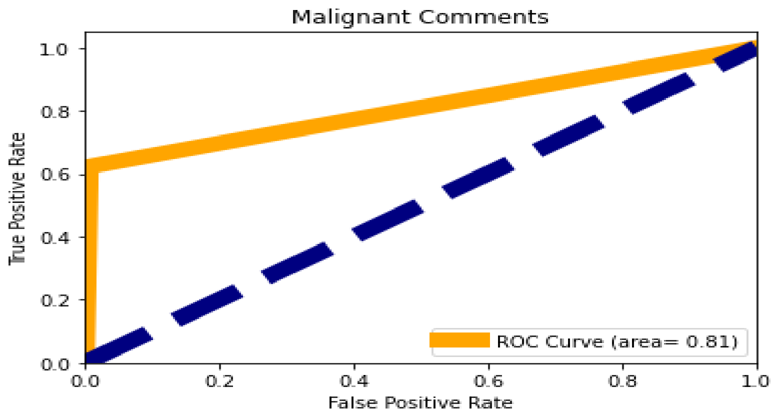


Here the final model gives 95% accuracy after tuning.

ROC-AUC Curve

```
fpr, tpr, threshold = roc_curve(y_test,pred)
auc = roc_auc_score(y_test,pred)
```

```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label="ROC Curve (area= %0.2f)" % auc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Malignant Comments")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So the best model has the area under curve as 0.81.

The Logistic Regression with RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV
param =       {'warm_start':[True,False],
             'dual':[True,False],
               'random_state':[50,70,100]}
```

```
rand_search = RandomizedSearchCV(lr,param_distributions=param,cv=2)
```

```
rand_search.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=2, estimator=LogisticRegression(),
                   param_distributions={'dual': [True, False],
                                        'random_state': [50, 70, 100],
                                        'warm_start': [True, False]})
```

```
rand_search.best_params_
```

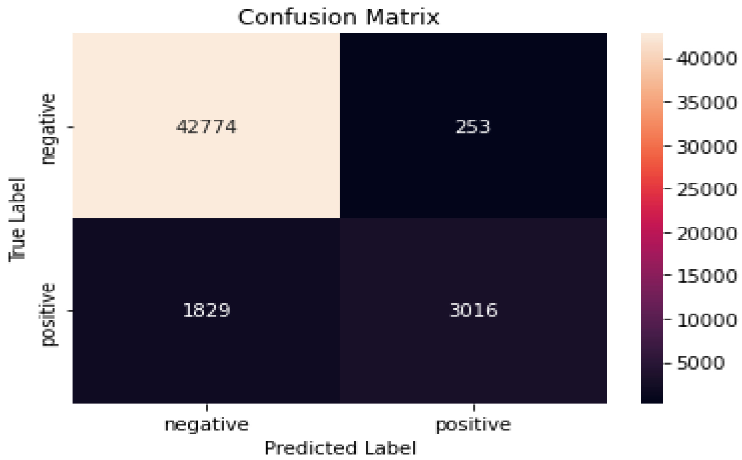
```
{'warm_start': True, 'random_state': 70, 'dual': False}
```

```
lr= LogisticRegression(warm_start=True,random_state=100,dual=False)
lr.fit(x_train,y_train)

y_pred1= lr.predict(x_test)
```

```
print(" Accuracy score :",accuracy_score(y_test,y_pred1),
      "\n","="*80,
      "\n Cross_validation_Score :", cross_val_score(lr,x,y,cv=5).mean(),
      "\n","="*80,
      "\n Classification report :\n",classification_report(y_test,y_pred1),
     "="*80,
      "\n Confusion matrix :\n",confusion_matrix(y_test,y_pred1))
```

```
 Accuracy score : 0.9565090240641712
 ================================================================================
 Cross_validation_Score : 0.9561637086494331
 ================================================================================
 Classification report :
               precision    recall  f1-score   support

           0       0.96      0.99      0.98     43027
           1       0.92      0.62      0.74      4845

    accuracy                           0.96     47872
   macro avg       0.94      0.81      0.86     47872
weighted avg       0.96      0.96      0.95     47872
 ================================================================================
 Confusion matrix :
 [[42774   253]
 [ 1829  3016]]
```

```
conf_mat = confusion_matrix(y_test, y_pred1)
class_label = ["negative", "positive"]
df = pd.DataFrame(conf_mat, index = class_label, columns = class_label)
sns.heatmap(df, annot = True,fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

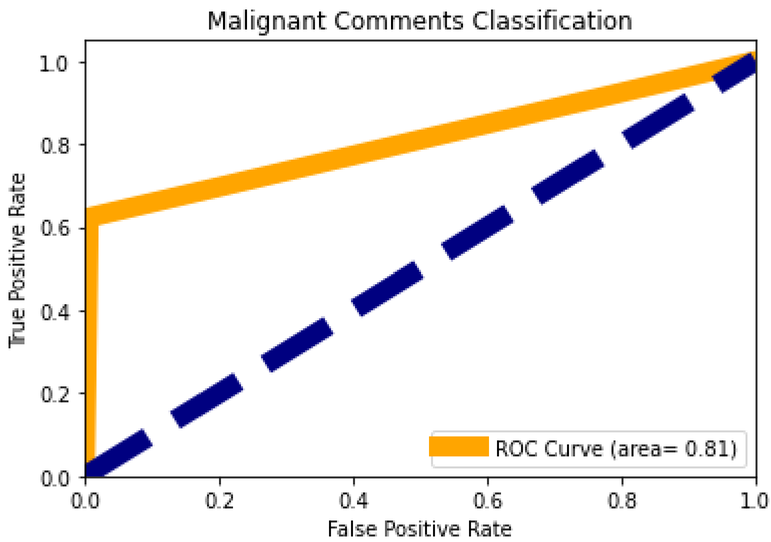


Here the final model gives 95% accuracy after tuning.

ROC-AUC Curve

```
fpr, tpr, threshold = roc_curve(y_test,y_pred1)
auc = roc_auc_score(y_test,y_pred1)
```

```
plt.figure()
plt.plot(fpr,tpr,color="orange",lw=10,label="ROC Curve (area= %0.2f)" % auc)
plt.plot([0,1],[0,1],color="navy",lw=10,linestyle="--")
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Malignant Comments Classification")
plt.legend(loc="lower right")
plt.show()
```



This is the AUC-ROC curve for the models which is plotted False positive rate against True positive rate. So the best model has the area under curve as 0.81.

We can see both method of hypertunning is giving same result. So, we can proceed with any one and here proceeding with The Logistic Regression with RandomizedSearchCV.

Saving The Predictive Model

```
filename='Malignant_Comments_Classification.pickle'
pickle.dump(lr,open(filename,'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
```

Comparing Actual and Predicted

```
a =np.array(y_test)
predicted=np.array(loaded_model.predict(x_test))
Malignant_Comments_Classification=pd.DataFrame({'Orginal':a,'Predicted':predicted}, index=range(len(a)))
Malignant_Comments_Classification
```

	Orginal	Predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1

Verifying Model on Testing Data

```
#test data (comments) converted to vectors
testing_data = tf_vec.fit_transform(cmnt_test["comment_text"])
```

```
prediction=lr.predict(testing_data)
prediction
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
cmnt_test['label'] = prediction
cmnt_test.head()
```

	comment_text	length	clean_cmnt_test	label
0	yo bitch rule succesful ever whats hating sad ...	367	221	0
1	rfc title fine imo	50	18	0
2	source zawe ashton lapland	54	26	0
3	look back source information updated correct f...	205	109	0
4	anonymously edit article	41	24	0

- <u>**Saving the model in CSV format**</u>

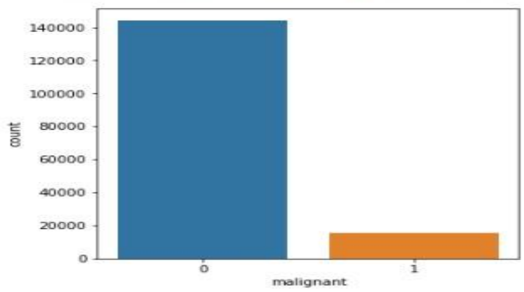
```
cmnt_test.to_csv('Malignant_Test.csv',index=False)
```

3. Key Metrics for success in solving problem under consideration
   - Accuracy Score, Precision Score, Recall Score, F1-Score and CV score are used for success. Also, confusion matrix and AUC-ROC Curve is used for success.

4. Visualization
   - Uni-Variate Analysis
     - Using Countplot

```
print(comments_train['malignant'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('malignant', data=comments_train)
```

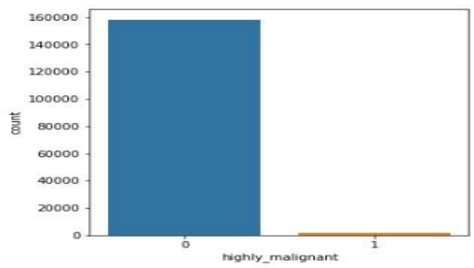
```
0    144277
1     15294
Name: malignant, dtype: int64
<AxesSubplot:xlabel='malignant', ylabel='count'>
```



- **We can observe that Total no of 15294 is Malignant Comment and Total no of 144277 is not Malignant Comment.**

```
print(comments_train['highly_malignant'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('highly_malignant', data=comments_train)
```

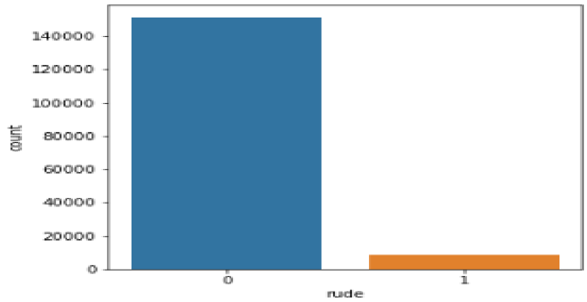
```
0    157976
1      1595
Name: highly_malignant, dtype: int64
<AxesSubplot:xlabel='highly_malignant', ylabel='count'>
```



- **We can observe that Total no of 1595 is highly_malignant Comment and Total no of 157976 is not highly_malignant Comment.**

```
print(comments_train['rude'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('rude', data=comments_train)
```

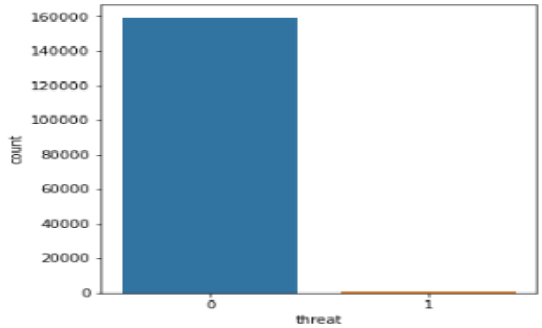
```
0    151122
1      8449
Name: rude, dtype: int64
<AxesSubplot:xlabel='rude', ylabel='count'>
```



- **We can observe that Total no of 8449 is rude Comment and Total no of 151122 is not rude Comment.**

```
print(comments_train['threat'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('threat', data=comments_train)
```

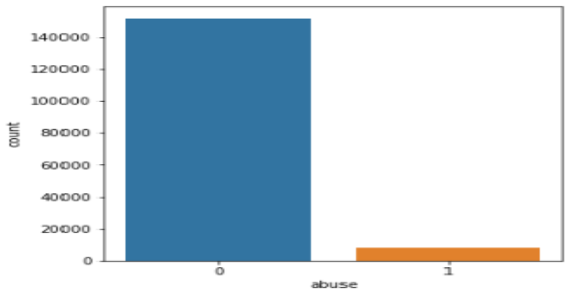
```
0    159093
1       478
Name: threat, dtype: int64
<AxesSubplot:xlabel='threat', ylabel='count'>
```



- **We can observe that Total no of 478 is threat Comment and Total no of 159093 is not threat Comment.**

```
print(comments_train['abuse'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('abuse', data=comments_train)
```

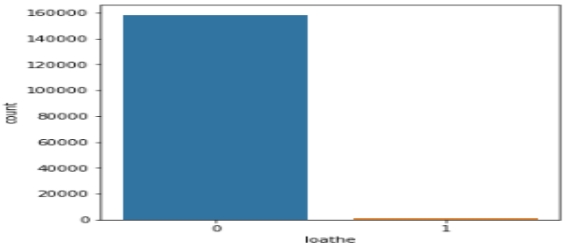
```
0    151694
1      7877
Name: abuse, dtype: int64
<AxesSubplot:xlabel='abuse', ylabel='count'>
```



- **We can observe that Total no of 7877 is abuse Comment and Total no of 151694 is not abuse Comment.**

```
print(comments_train['loathe'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot('loathe', data=comments_train)
```

```
0     158166
1       1405
Name: loathe, dtype: int64
<AxesSubplot:xlabel='loathe', ylabel='count'>
```



- **We can observe that Total no of 1405 is loathe Comment and Total no of 158166 is not loathe Comment.**

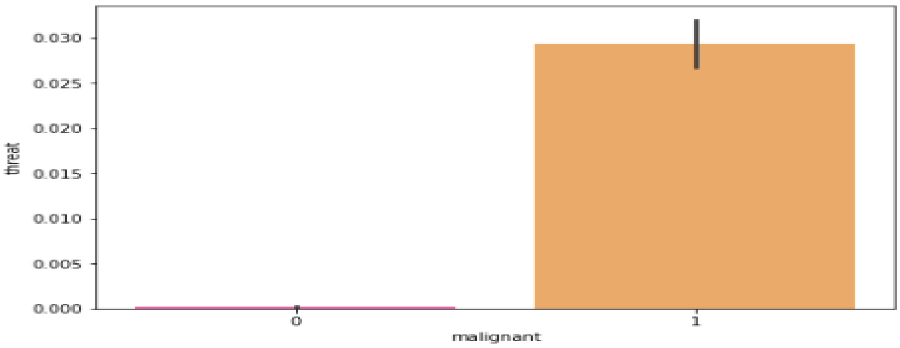
- Bivariate Analysis

(For comparison between each feature with target)

- Using Bar plot

```
plt.figure(figsize=(8,5))
sns.barplot(x='malignant',y='threat',data= comments_train, palette='spring')
```

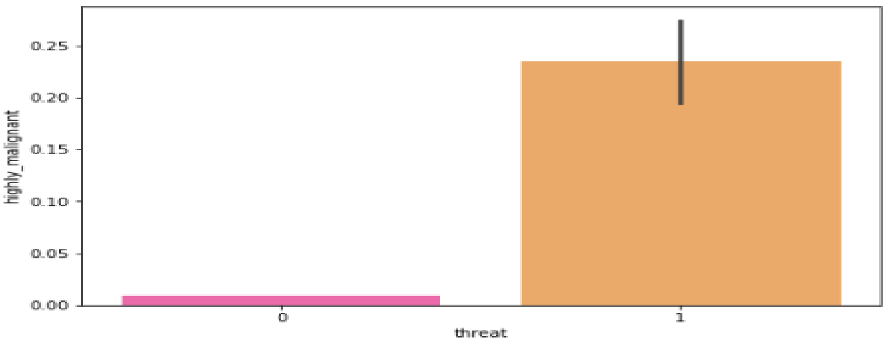
```
<AxesSubplot:xlabel='malignant', ylabel='threat'>
```



- **We can observe that malignant with threat Comment is most.**

```
plt.figure(figsize=(8,5))
sns.barplot(x='threat',y='highly_malignant',data= comments_train, palette='spring')
```

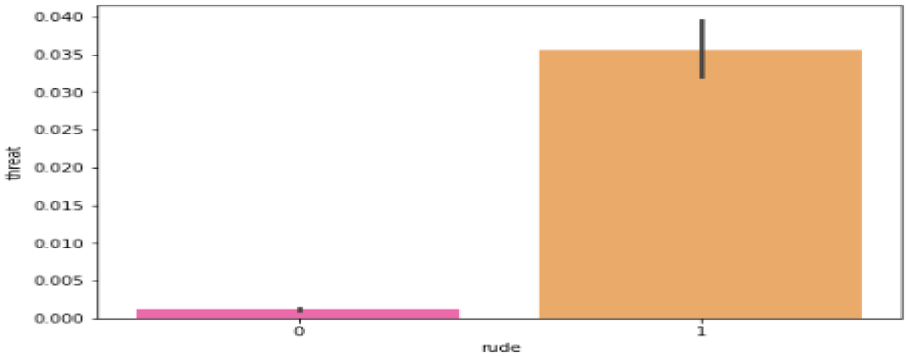
```
<AxesSubplot:xlabel='threat', ylabel='highly_malignant'>
```



- **We can observe that highly_malignant with threat Comment is most.**

```
plt.figure(figsize=(8,5))
sns.barplot(x='rude',y='threat',data= comments_train, palette='spring')
```

```
<AxesSubplot:xlabel='rude', ylabel='threat'>
```

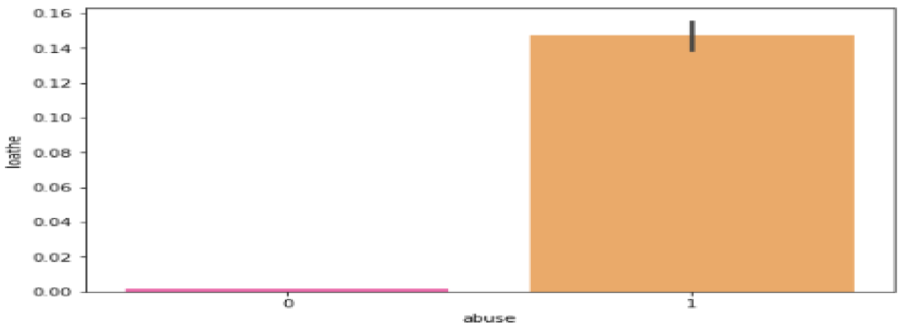


- **We can observe that rude with threat comment is most.**

- **We can observe that rude with threat comment is most.**


```
plt.figure(figsize=(8,5))
sns.barplot(x='abuse',y='loathe',data= comments_train, palette='spring')
```

```
<AxesSubplot:xlabel='abuse', ylabel='loathe'>
```



- **We can observe that abuse with loathe comment is most.**

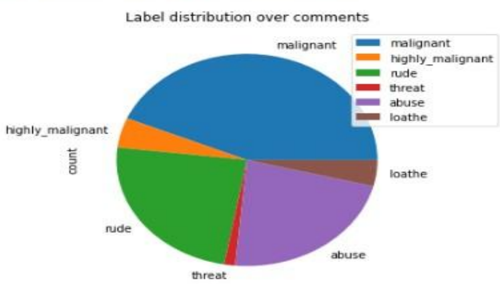
- Multivariate Analysis

  (For comparison between all features with target)

  - Using Pie-Plot

```
#checking how which comment fall under which category
cols = ['malignant','highly_malignant','rude','threat','abuse','loathe']
df_plot = comments_train[cols].sum().to_frame().rename(columns={0: 'count'})
df_plot.plot.pie(y='count',title='Label distribution over comments',figsize=(5, 5)).legend(loc='center left', bbox_to_anchor=(0.8, 0.8))
```

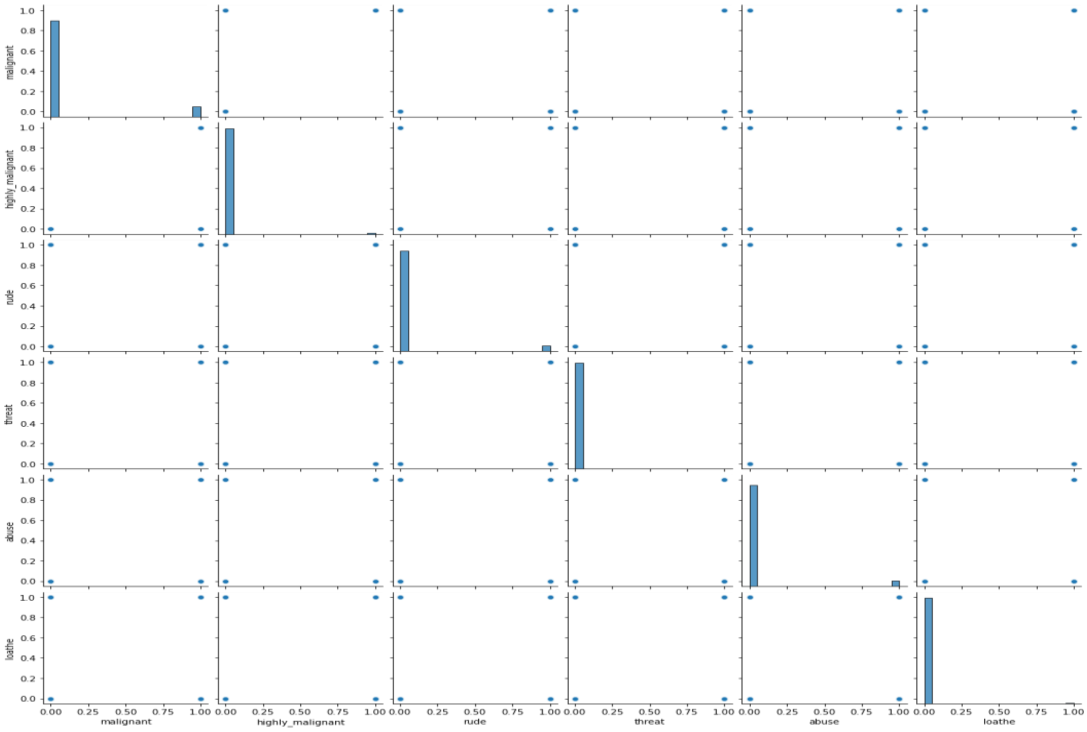
```
<matplotlib.legend.Legend at 0x27b98ba0d60>
```



- **We can observe that total distribution of all comment's type in which we can see malignant comments are most and threat comments are least.**

➢ Using Pair plot

```
sns.pairplot(comments_train)
```



5. Interpretation of the Results

- Through Pre-processing it is interpretive Converted all messages to lower case, Replaced email addresses with 'email', Replaced URLs with 'webaddress', Replaced money symbols with 'moneysymb', (£ can by typed with ALT key + 156), Replaced 10digit phone numbers (formats include parenthesis, spaces, no spaces, dashes) with 'phone number', Replace Numbers with 'number', Removed Punctuation, Replaced extra space, Replaced leading and trailing white space, Removed \n, Added and removed stop words, Calculated length of sentence, Made one Target Column, Removed Total length, Converted text into vectors using TF-IDF

- By creating/building model we get best model: Logistic Regression.

CONCLUSION

1. Key Findings and Conclusions of the Study

Here we have made a MALIGNANT COMMENTS CLASSIFICATION. We have done EDA, cleaned data and Visualized Data. While cleaning the data it is analyzed that:

- One column "id" is irrelevant so dropped this column.

After that we have done prediction on basis of Data using Data Pre-processing, Checked Correlation, removed email addresses, URLs, money symbols, 10digit phone numbers, Punctuation, extra space, leading and trailing white space, \n, stop words, converted text into vectors using TF-IDF and at last train our data by splitting our data through train-test split process.

Built our model using 7 models and finally selected best model which was giving best accuracy that is Logistic Regression. Then tuned our model through Hyper Tuning using GridSearchCV and RandomizedSearchCV, in which proceeded with RandomizedSearchCV. And at last compared our predicted and Actual test data. Thus, our project is completed.

2. Learning Outcomes of the Study in respect of Data Science

- This project has demonstrated the importance of NLP.
- Through different powerful tools of visualization, we were able to analyze and interpret the huge data and with the help of pie plot, count plot & word cloud, I am able to see the distribution of threat comments.
- Through data cleaning we were able to remove unnecessary columns, values, special characters, symbols, stop-words and punctuation from our dataset due to which our model would have suffered from over fitting or under fitting.

The few challenges while working on this project were: -

- To find punctuations & stop words, which took time to run using NLP.
- The data set is huge it took time to run some algorithms & to check the cross-validation score.

3. Limitations of this work and Scope for Future Work

While we couldn't reach out goal of 100% accuracy but created a system that made data get very close to that goal. This project allows multiple algorithms to be integrated together to combine modules and their results to increase the accuracy of the final result.

For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset.