MALIGNANT COMMENTS CLASSIFICATION PROJECT

Submitted by

NARRALA HOMAKIRAN

ACKNOWLEDGMENT

The project entitled "MALIGNANT COMMENTS CLASSIFICATION" is done by me during my internship with Flip Robo Technologies. I am grateful to Data Trained and Flip Robo Technologies for their guidance during this project. Other reference websites used to complete this project are:

1. Data source provided by the client.
2. Wikipedia
3. Towardsdatascience.com
4. Datacamp.com

INTRODUCTION

Conceptual Background of the Domain Problem

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

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

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

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

PROBLEM STATEMENT:

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

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

- With continuous increase in available data, there is a pressing need to organize it and modern classification problems often involve the prediction of multiple labels simultaneously associated with a single instance. Known as Multi-Label Classification, it is one such task which is omnipresent in many real world problems. In this project also, we have multi-label classification problem.

- We have used Tf-Idf Vectorizer to vectorize the words in our dataset. TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. It is very important for tuning performance on NLP projects.

- The TF-IDF score for the word t in the document d from the document set D is calculated as follows:

$$tf\,idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

- Where:

$$tf(t, d) = \log(1 + freq(t, d))$$

$$idf(t, D) = \log\left(\frac{N}{count(d \in D : t \in d)}\right)$$

Data Sources and their formats

- The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes 'Id', 'Comments', 'Malignant', 'Highly malignant', 'Rude', 'Threat', 'Abuse' and 'Loathe'.
- The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes:

- **Malignant:** It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
- **Highly Malignant:** It denotes comments that are highly malignant and hurtful.
- **Rude:** It denotes comments that are very rude and offensive.
- **Threat:** It contains indication of the comments that are giving any threat to someone.

- **Abuse:** It is for comments that are abusive in nature.
- **Loathe:** It describes the comments which are 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.

The sample data for the reference is as shown below:

Out[4]:

	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
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0

```
In [5]: df_test.head(10)
```

Out[5]:

	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.
5	0001ea8717f6de06	Thank you for understanding. I think very high...
6	00024115d4cbde0f	Please do not add nonsense to Wikipedia. Such ...
7	000247e83dcc1211	:Dear god this site is horrible.
8	00025358d4737918	" \n Only a fool can believe in such numbers. ...
9	00026d1092fe71cc	== Double Redirects == \n\n When fixing double...

- Then we further checked more about data using info, shapes using .shape, columns using .columns(),null values using .isnull. .sum().sum(), and further visualize it through heatmap as follows:

```
In [6]: df_train.shape
Out[6]: (159571, 8)
```

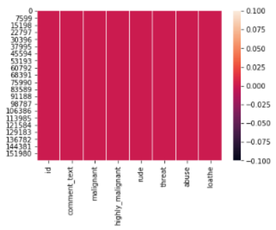
```
In [7]: df_test.shape
Out[7]: (153164, 2)
```

```
In [8]: df_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
```

```
In [9]: df_train.columns
Out[9]: Index(['id', 'comment_text', 'malignant', 'highly_malignant', 'rude', 'threat',
       'abuse', 'loathe'],
      dtype='object')
```

```
In [10]: df_train.isnull().sum()
Out[10]: id                  0
         comment_text        0
         malignant           0
         highly_malignant    0
         rude                0
         threat              0
         abuse               0
         loathe              0
         dtype: int64
```

```
In [11]: sns.heatmap(df_train.isnull())
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2d84d041730>
```



- Then we perform Exploratory Data Analysis(EDA) as follows:

```
In [12]: # Dropping 'id' to reduce unnecessary feature
         df_train.drop(['id'],axis=1,inplace=True)
```

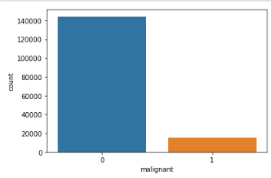
```
In [13]: df_train.describe()
Out[13]:
```

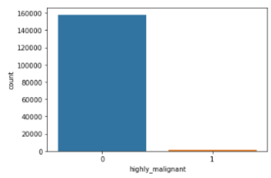
	malignant	highly_malignant	rude	threat	abuse	loathe
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420
min	0.000000	0.000000	0.000000	0.000000	0.000000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

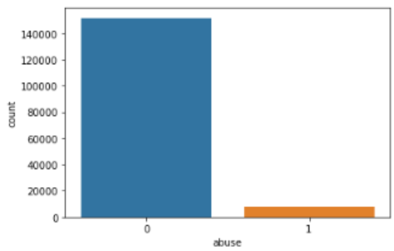
```
In [14]: features=df_train.columns[1:]
         features
Out[14]: Index(['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe'], dtype='object')
```

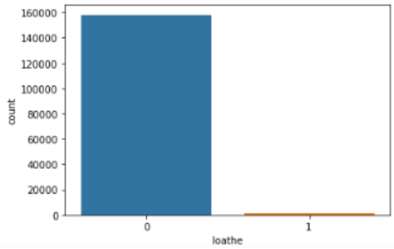
```
In [15]: for i in features:
             print('Number of unique values in {} : {}'.format(i, df_train[i].nunique()))

         Number of unique values in malignant : 2
         Number of unique values in highly_malignant : 2
         Number of unique values in rude : 2
         Number of unique values in threat : 2
         Number of unique values in abuse : 2
         Number of unique values in loathe : 2
```

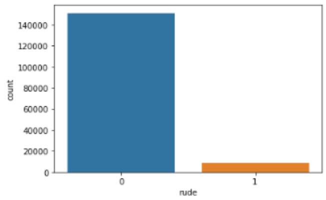
```
In [16]: for i in features:
             sns.countplot(df_train[i])
             plt.show()
```

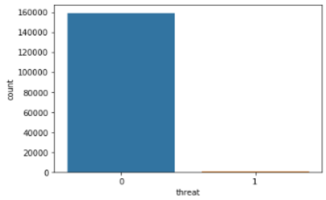












In [17]:
```
for i in features:
    print('Number of unique values in {} : {}'.format(i, df_train[i].value_counts()))
```

```
Number of unique values in malignant : 0    144277
1     15294
Name: malignant, dtype: int64
Number of unique values in highly_malignant : 0    157976
1      1595
Name: highly_malignant, dtype: int64
Number of unique values in rude : 0    151122
1      8449
Name: rude, dtype: int64
Number of unique values in threat : 0    159093
1       478
Name: threat, dtype: int64
Number of unique values in abuse : 0    151694
1      7877
Name: abuse, dtype: int64
Number of unique values in loathe : 0    158166
1      1405
Name: loathe, dtype: int64
```

In [18]:
```
good_comments = df_train[(df_train['malignant']!=1) & (df_train['highly_malignant']!=1) & (df_train['rude']!=1) &
                         (df_train['threat']!=1) & (df_train['abuse']!=1) & (df_train['loathe']!=1)]
good_percent=len(good_comments)/len(df_train)*100
print('Percentage of good comments = ',good_percent)
print('Percentage of negative comments = ', (100-good_percent))
```

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

```
In [19]: df_train['comment_length']=df_train.comment_text.str.len()
         df_train
```

Out[19]:

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	comment_length
0	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264
1	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112
2	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	233
3	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	622
4	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	67
...	...	...	...	...	...	...	...	...
159566	":::::And for the second time of asking, when ...	0	0	0	0	0	0	295
159567	You should be ashamed of yourself \n\nThat is ...	0	0	0	0	0	0	99
159568	Spitzer \n\nUmm, theres no actual article for ...	0	0	0	0	0	0	81
159569	And it looks like it was actually you who put ...	0	0	0	0	0	0	116
159570	"\nAnd ... I really don't think you understand...	0	0	0	0	0	0	189

159571 rows × 8 columns

```
In [20]: df_test['comment_length']=df_test.comment_text.str.len()
         df_test.head()
```

Out[20]:

	id	comment_text	comment_length
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...	367
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...	50
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	54
3	00017563c3f7919a	:If you have a look back at the source, the in...	205
4	00017695ad8997eb	I don't anonymously edit articles at all.	41

```
In [21]: # Creating a new feature having Negative Comments and Non-Negative Comments.
         df_train['label'] = df_train[features].max(axis=1)
         df_train.head(10)
```

Out[21]:

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	comment_length	label
0	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264	0
1	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112	0
2	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	233	0
3	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	622	0
4	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	67	0
5	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0	65	0
6	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0	44	1
7	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0	115	0
8	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0	472	0
9	alignment on this subject and which are contra...	0	0	0	0	0	0	70	0

```
In [22]: df_train['label'].value_counts()

Out[22]: 0    143346
         1     16225
         Name: label, dtype: int64
```

- Next we will be using a function to filter using POS tagging. Also, all the pre-processing steps needed to clean the data.

```
In [24]: def get_pos(pos_tag):
             if pos_tag.startswith('J'):
                 return wordnet.ADJ
             elif pos_tag.startswith('N'):
                 return wordnet.NOUN
             elif pos_tag.startswith('R'):
                 return wordnet.ADV
             else:
                 return wordnet.NOUN
```

```
In [25]: # Function for data cleaning...
         def Processed_data(comments):
             # Replace email addresses with 'email'
             comments=re.sub(r'^.+@[^\.].*\.[a-z]{2,}$',' ', comments)

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

             # getting only words(i.e removing all the special characters)
             comments = re.sub(r'[^\w]', ' ', comments)

             # getting only words(i.e removing all the" _ ")
             comments = re.sub(r'[\_]', ' ', comments)

             # getting rid of unwanted characters(i.e remove all the single characters left)
             comments=re.sub(r'\s+[a-zA-Z]\s+', ' ', comments)

             # Removing extra whitespaces
             comments=re.sub(r'\s+', ' ', comments, flags=re.I)

             #converting all the letters of the review into lowercase
             comments = comments.lower()

             # splitting every words from the sentences
             comments = comments.split()

             # iterating through each words and checking if they are stopwords or not,
             comments=[word for word in comments if not word in set(STOPWORDS)]

             # remove empty tokens
             comments = [text for text in comments if len(text) > 0]

             # getting pos tag text
             pos_tags = pos_tag(comments)

             # considering words having length more than 3only
             comments = [text for text in comments if len(text) > 3]

             # performing lemmatization operation and passing the word in get_pos function to get filtered using POS ...
             comments = [(WordNetLemmatizer().lemmatize(text[0], get_pos(text[1])))for text in pos_tags]

            # considering words having length more than 3 only
             comments = [text for text in comments if len(text) > 3]
             comments = ' '.join(comments)
             return comments
```

- After performing all the above steps and also adding a new feature to check new comment length after cleaning, our dataset would look as follows:

```
In [28]: # Adding new feature clean_comment_length to store Length of characters
df_train['clean_comment_length'] = df_train['clean_comment_text'].apply(lambda x: len(str(x)))
df_train.head()
```

Out[28]:

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	comment_length	label	clean_comment_text	clean_comment_length
0	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264	0	explanation edits username hardcore metallica ...	129
1	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112	0	match background colour seemingly stuck thanks...	64
2	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	233	0	trying edit constantly removing relevant infor...	112
3	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	622	0	real suggestion improvement wondered section s...	315
4	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	67	0	hero chance remember page	25

```
In [29]: # Adding new feature clean_comment_length to store Length of characters
df_test['clean_comment_length'] = df_test['clean_comment_text'].apply(lambda x: len(str(x)))
df_test.head()
```

Out[29]:

	id	comment_text	comment_length	clean_comment_text	clean_comment_length
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...	367	bitch rule succesful whats hating mofuckas bit...	184
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...	50	title fine	10
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	54	source zawe ashton lapland	26
3	00017563c3f7919a	:If you have a look back at the source, the in...	205	look source information updated correct form g...	109
4	00017695ad8997eb	I don't anonymously edit articles at all.	41	anonymously edit article	24

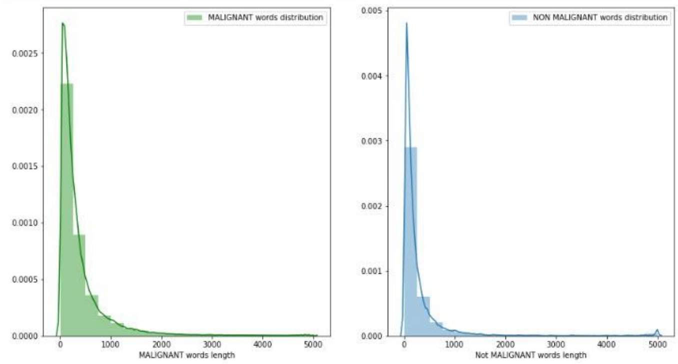
- We have also observed most frequent words in positive and negative comments through word-cloud:

```
In [31]: # Non-Negative Comments - Frequent words in training data
Display_wordcloud(df_train['clean_comment_text'][df_train['label']==0],"Positive Comments")
```

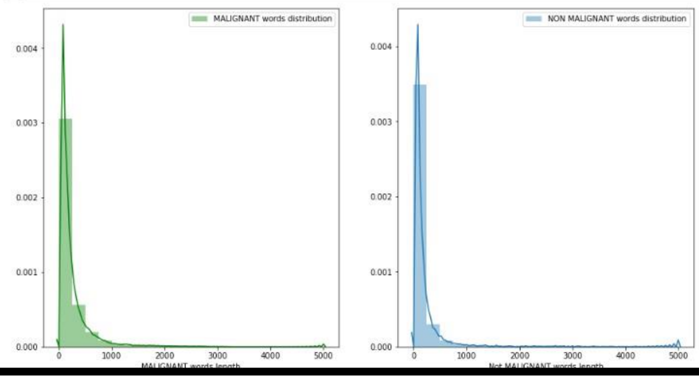

```
In [32]: # Non-Negative Comments - Frequent words in training data
Display_wordcloud(df_train['clean_comment_text'][df_train['label']==1],"Positive Comments")
```

- Then we have checked distribution of comment length before and after cleaning.

```
In [33]: # Comments length distribution BEFORE cleaning
f,ax = plt.subplots(1,2,figsize = (15,8))

sns.distplot(df_train[df_train['label']==0]['comment_length'],bins=20,ax=ax[0],label='MALIGNANT words distribution',color='g')

ax[0].set_xlabel('MALIGNANT words length')
ax[0].legend()

sns.distplot(df_train[df_train['label']==1]['comment_length'],bins=20,ax=ax[1],label='NON MALIGNANT words distribution')
ax[1].set_xlabel('Not MALIGNANT words length')
ax[1].legend()

plt.show()
```




```
In [34]: # Comments length distribution after cleaning
f,ax = plt.subplots(1,2,figsize = (15,8))

sns.distplot(df_train[df_train['label']==0]['clean_comment_length'],bins=20,ax=ax[0],label='MALIGNANT words distribution',color=

ax[0].set_xlabel('MALIGNANT words length')
ax[0].legend()

sns.distplot(df_train[df_train['label']==1]['clean_comment_length'],bins=20,ax=ax[1],label='NON MALIGNANT words distribution')
ax[1].set_xlabel('Not MALIGNANT words length')
ax[1].legend()

plt.show()
```



Preparing Data For Modelling

- We are using TF-IDF vectorizer for vectorizing the words.

```
In [35]: # TF-IDF(term frequency-inverse document frequency) vectorizer
def Tf_idf_train(text):
    tfid = TfidfVectorizer(min_df=3,smooth_idf=False)
    return tfid.fit_transform(text)

In [36]: # Let's define x, y for modelling
x=Tf_idf_train(df_train['clean_comment_text'])
x.shape

Out[36]: (159571, 43194)

In [37]: # For y
y = df_train['label'].values
y.shape

Out[37]: (159571,)
```

MODEL BUILDING

```
In [38]: # Importing useful libraries for model training

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier


# Model selection libraries...
from sklearn.model_selection import cross_val_score, cross_val_predict, train_test_split
from sklearn.model_selection import GridSearchCV


# Importing some metrics we can use to evaluate our model performance....
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
In [39]: #splitting the data into training and testing
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_size=0.30,stratify=y)
```

```
In [40]: # Creating instances for different Classifiers

LR=LogisticRegression()
MNB=MultinomialNB()
DT=DecisionTreeClassifier()
KNN=KNeighborsClassifier()
SV=SVC()
```

```
In [41]: #     Putting Scikit-Learn machine Learning Models in a List so that it can be used for further evaluation in loop.
models=[]
models.append(('LogisticRegression',LR))
models.append(('MultinomialNB',MNB))
models.append(('DecisionTreeClassifier',DT))
models.append(('KNeighborsClassifier',KNN))
models.append(('SVC',SV))
```

- We obtained following results after training the model on various algorithms:

```
************** LogisticRegression ***************


LogisticRegression()


Accuracy_score= 0.9531667780748663


Cross_Val_Score= 0.9535128562209592


roc_auc_score= 0.7924918146002661


classification report
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     43004
           1       0.92      0.59      0.72      4868

    accuracy                           0.95     47872
   macro avg       0.94      0.79      0.85     47872
weighted avg       0.95      0.95      0.95     47872


[[42754   250]
 [ 1992  2876]]


AxesSubplot(0.125,0.808774;0.62x0.0712264)
```

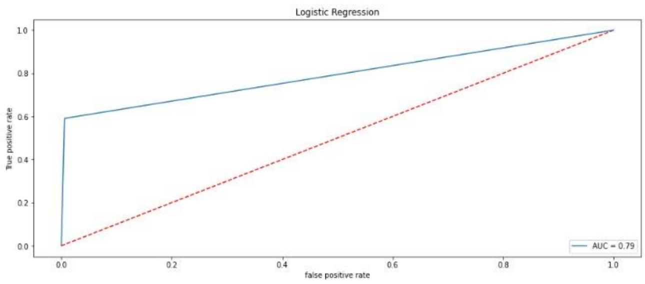
```
************** MultinomialNB ***************

MultinomialNB()

Accuracy_score= 0.9354737633689839

Cross_Val_Score= 0.9367303554748633

roc_auc_score= 0.6884622511658735

classification report
              precision    recall  f1-score   support

           0       0.93      1.00      0.97     43004
           1       0.97      0.38      0.54      4868

    accuracy                           0.94     47872
   macro avg       0.95      0.69      0.75     47872
weighted avg       0.94      0.94      0.92     47872


[[42941    63]
 [ 3026  1842]]

AxesSubplot(0.125,0.808774;0.62x0.0712264)
```

```
************** DecisionTreeClassifier ***************

DecisionTreeClassifier()

Accuracy_score= 0.9392337901069518

Cross_Val_Score= 0.9399076251956352

roc_auc_score= 0.8263624575788062

classification report
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     43004
           1       0.71      0.68      0.70      4868

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


[[41630  1374]
 [ 1535  3333]]

AxesSubplot(0.125,0.808774;0.62x0.0712264)
```

```
************** KNeighborsClassifier ***************

KNeighborsClassifier()

Accuracy_score= 0.8963277072192514

Cross_Val_Score= 0.8967356214680471

roc_auc_score= 0.6214955391587276

classification report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94     43004
           1       0.48      0.28      0.35      4868

    accuracy                           0.90     47872
   macro avg       0.70      0.62      0.65     47872
weighted avg       0.88      0.90      0.88     47872


[[41563  1441]
 [ 3522  1346]]

AxesSubplot(0.125,0.808774;0.62x0.0712264)


************** SVC ***************

SVC()

Accuracy_score= 0.9545872326203209
```

- Since the dataset was too large it took me around 9-10 hours to get these results for these algorithms. Hence I had to interrupt the kernel and proceed with available results. Out of all the algorithms, Logistic Regression was giving best score. Also, its cross validation score was also satisfactory. Its ROC_AUC curve is as shown:

```
In [47]: LR.fit(x_train,y_train)
         LR.score(x_train,y_train)
         pred=LR.predict(x_test)
         print('Accuracy Score:',accuracy_score(y_test,pred))
         print('Confusion Matrix:',confusion_matrix(y_test,pred))
         print('Classification Report:','\n',classification_report(y_test,pred))

         Accuracy Score: 0.9531667780748663
         Confusion Matrix: [[42754   250]
          [ 1992  2876]]
         Classification Report:
                       precision    recall  f1-score   support

                    0       0.96      0.99      0.97     43004
                    1       0.92      0.59      0.72      4868

             accuracy                           0.95     47872
            macro avg       0.94      0.79      0.85     47872
         weighted avg       0.95      0.95      0.95     47872

Out[56]: Text(0.5, 1.0, 'Logistic Regression')
```



PREDICTING TEST DATASET

```
In [51]: def Tf_idf_test(text):
             tfid = TfidfVectorizer(max_features=43194,smooth_idf=False)
             return tfid.fit_transform(text)

In [52]: x_test_data=Tf_idf_test(df_test['clean_comment_text'])

In [53]: x_test_data.shape

Out[53]: (153164, 43194)

In [54]: Prediction=LR.predict(x_test_data)
         df_test['Predicted Labels']=Prediction
         df_test

Out[54]:
```

	id	comment_text	comment_length	clean_comment_text	clean_comment_length	Predicted Labels
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...	367	bitch rule succesful whats hating mofuckas bit...	184	0
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...	50	title fine	10	0
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	54	source zawe ashton lapland	26	0
3	00017563c3f7919a	:If you have a look back at the source, the in...	205	look source information updated correct form g...	109	0
4	00017695ad8997eb	I don't anonymously edit articles at all.	41	anonymously edit article	24	0
...	...	...	...	...	...	...
153159	fffcd0960ee309b5	. \n i totally agree, this stuff is nothing bu...	60	totally agree stuff long crap	29	0
153160	fffd7a9a6eb32c16	== Throw from out field to home plate. == \n\n...	198	throw field home plate faster throwing direct ...	85	0
153161	fffda9e8d6fafa9e	" \n\n == Okinotorishima categories == \n\n I ...	423	okinotorishima category change agree correct g...	212	0

```
In [48]: df_test['Predicted Labels'].value_counts()

Out[48]: 0    152452
         1       712
         Name: Predicted Labels, dtype: int64

In [49]: # Pickle file.
         import joblib
         joblib.dump(LR,'Malignant_Prediction.pkl')

Out[49]: ['Malignant_Prediction.pkl']

In [50]: df_test.to_csv('Malignant_Predict.csv')
```

CONCLUSION

- We have got Logistic Regression as best model since it’s giving us good result and other metrics are also satisfactory.
- Using Logistic Regression as our final algorithm we have predicted the values for test dataset and it’s also working well and is able to differentiate/predict negative comments and non-negative (good) comments.