FLIP ROBO

Malignant-Comments-Classifier Project

Submitted By:

Suraj Kumar Soni

(Internship Batch : 29)

Acknowledgment

I would like to express my sincere thanks and gratitude to my SME Mr. Shwetank Mishra as well as the “FlipRobo Technologies” team for letting me work on the “Used Car Price Prediction” project. Their suggestions and directions have helped me in the completion of this project successfully. This project also helped me in doing lots of research wherein I came to know about so many new things.

Introduction

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.

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:-

In this project, we will develop and evaluate the performance and predictability of trained and tested models based on comments which is provide by flip robo techknology. Once we get a good fit, we will apply on our test data.

In here we will use various classification algorithm to predict our target. Let's have an overview of the algorithms we will use for our predictions. To read more about these algorithms , just click on the algorithms name.

- LogisticRegression:- Logistic regression analysis is valuable for predicting the likelihood of an event. It helps determine the probabilities between any two classes. In a nutshell, by looking at historical data, logistic regression can predict whether: An email is a spam.

- ❖ DecisionTreeClassifier:- Decision trees **help you to evaluate your options**. Decision Trees are excellent tools for helping you to choose between several courses of action. They provide a highly effective structure within which you can lay out options and investigate the possible outcomes of choosing those options.

- ❖ SVR:- The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points. Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value.

- ❖ KNeighborsClassifier:- By default, the KNeighborsClassifier looks for the 5 nearest neighbors. We must explicitly tell the classifier to use Euclidean distance for determining the proximity between neighboring points. Using our newly trained model, we predict whether a tumor is benign or not given its mean compactness and area..
- ❖ RandomForestClassifier:- What is Randomforestclassifier in Python? A random forest classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting..

Data Sources and their formats:- The dataset which I use for model making is provide by FlipRobo Technology, The data set has 159571 rows and 8 columns.

Dataset looks as follows:-

```
df
```

	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
...	...	...	...	...	...	...	...	...
159566	ffe987279560d7ff	":::::And for the second time of asking, when ...	0	0	0	0	0	0
159567	ffea4adeee384e90	You should be ashamed of yourself \n\nThat is ...	0	0	0	0	0	0
159568	ffee36eab5c267c9	Spitzer \n\nUmm, theres no actual article for ...	0	0	0	0	0	0
159569	fff125370e4aaaf3	And it looks like it was actually you who put ...	0	0	0	0	0	0
159570	fff46fc426af1f9a	"\nAnd ... I really don't think you understand...	0	0	0	0	0	0

159571 rows × 8 columns

About The DataSet:-

Variable	Definition
id	A unique id aligned with each comment text.
comment_text	It includes the comment text.
malignant	It is a column with binary values depicting which comments are malignant in nature.
highly_malignant	Binary column with labels for highly malignant text.
rude	Binary column with labels for comments that are rude in nature.
threat	Binary column with labels for threatening context in the comments.
abuse	Binary column with labels with abusive behaviour.
loathe	Label to comments that are full of loathe and hatred.

Data Preprocessing Done:-

For the purpose of the project the dataset has been preprocessed as follows:

- Checking shape of the dataframe

- Checking Missing Value
- Checking which type of data stored in each columns
- Text processing
- Plot Word cloud
- Visualization
- Describing the dataset
- Checking correlation and using heatmap for better understanding

We'll now open a python 3 Jupyter Notebook and execute the following code snippet to load the dataset and remove the non-essential features. Recieving a success message if the actions were correclty performed.

Hardware and Software Requirements and Tools Used:-

- *Hardware:-* ***Desktop/Laptop***
- *Software:-* ***Anaconda***
- *Libraries:-* **Numpy,Pandas,Matplot,Seaborn,nltk,etc**

Model/s Development and Evaluation

- ## *Identification of possible problem-solving approaches (methods):*

➢ Missing Value Handling:

There are 2 primary ways of handling missing values:

- ❖ Deleting the Missing values:-Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values.

- ❖ Imputing the Missing Values:- There are different ways of replacing the missing values
  - ✓ Replacing With Mean
  - ✓ Replacing With Mode
  - ✓ Replacing With Median,etc.
  - ✓ We are free from missing value otherwise it is very important step for model building

• Data Input Output Logic with WordCloud:-

I have analysed yhe input output logic with word cloud and I have word clouded the sentenced that as classified as foul language in every category. A tag/word cloud is a novelty visual representation of text data,typically used to depict keyword metadata on websites, or to visualize free from text. It's an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency or importance.

Code:-

```python
def wcloud(df, label):

    # lets print only rows where the label value is 1 (ie. where comment is harsh)
    subset=df[df[label]==1]
    text=subset.comment_text.values
    wc= WordCloud(background_color="black",max_words=4500)

    wc.generate(" ".join(text))

    plt.figure(figsize=(27,27))
    plt.subplot(221)
    plt.axis("off")
    plt.title("Words frequented in {}".format(label), fontsize=18)
    plt.imshow(wc.recolor(colormap= 'gist_earth' , random_state=244))
```

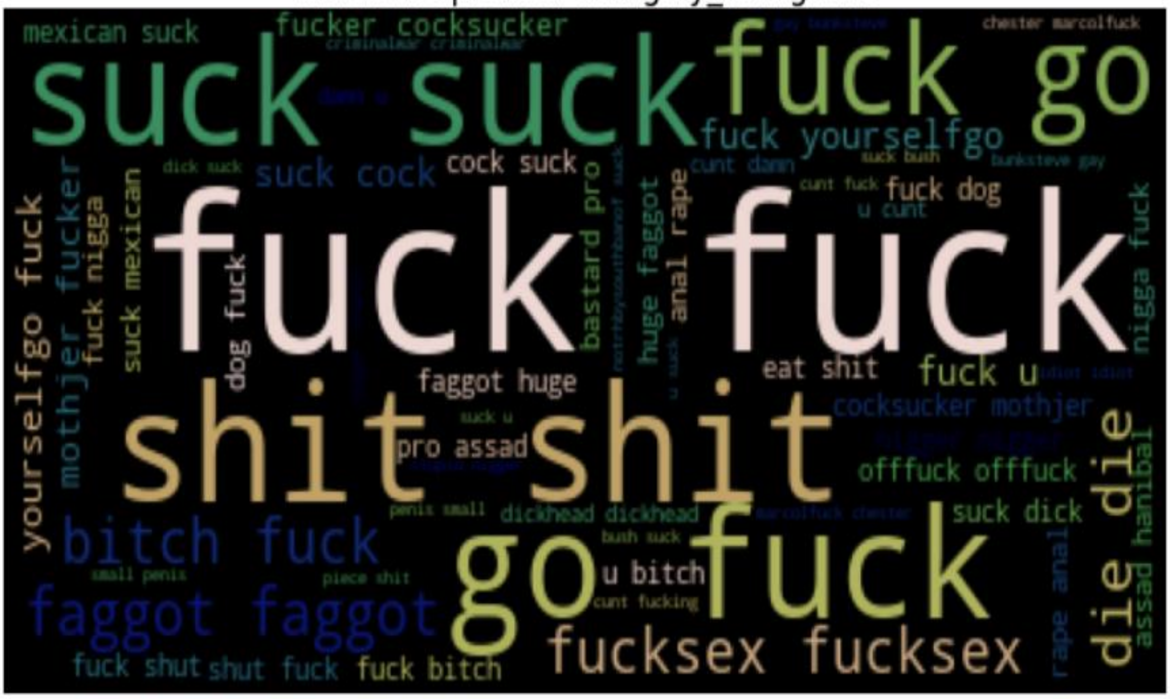
Output:-

```
df_m=df.loc[:,['comment_text','malignant']]
wcloud(df_m,'malignant')
```

Words frequented in malignant

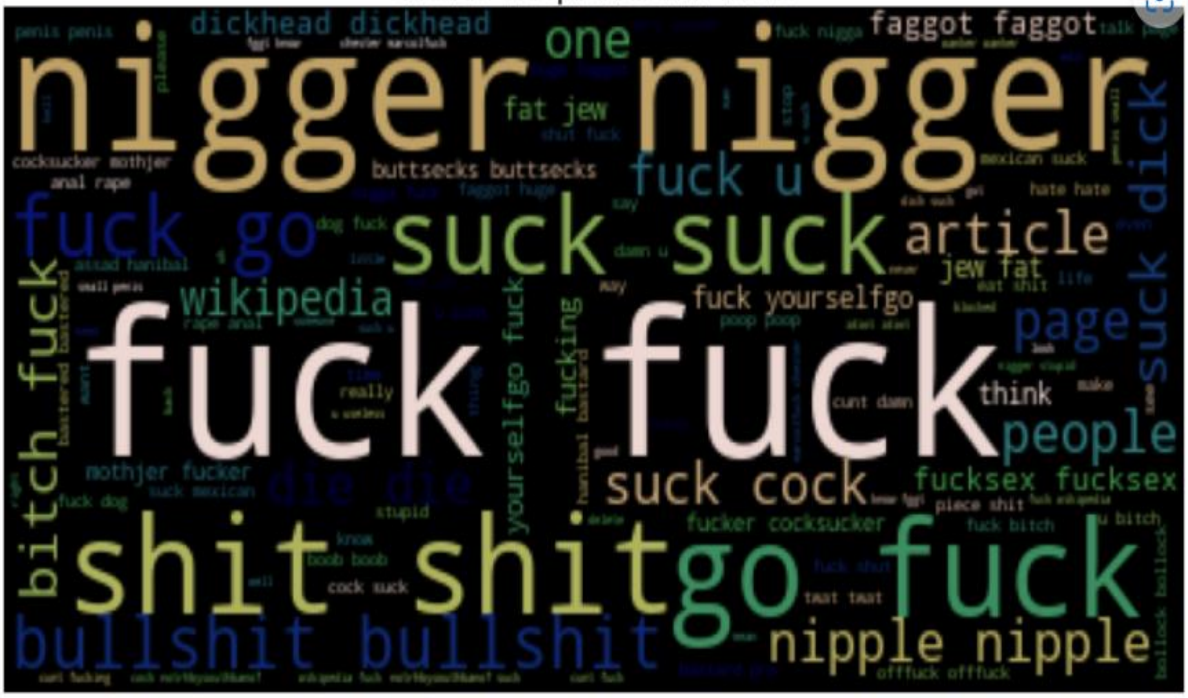

```
df_hm=df.loc[:,['comment_text','highly_malignant']]
wcloud(df_hm,'highly_malignant')
```

Words frequented in highly_malignant



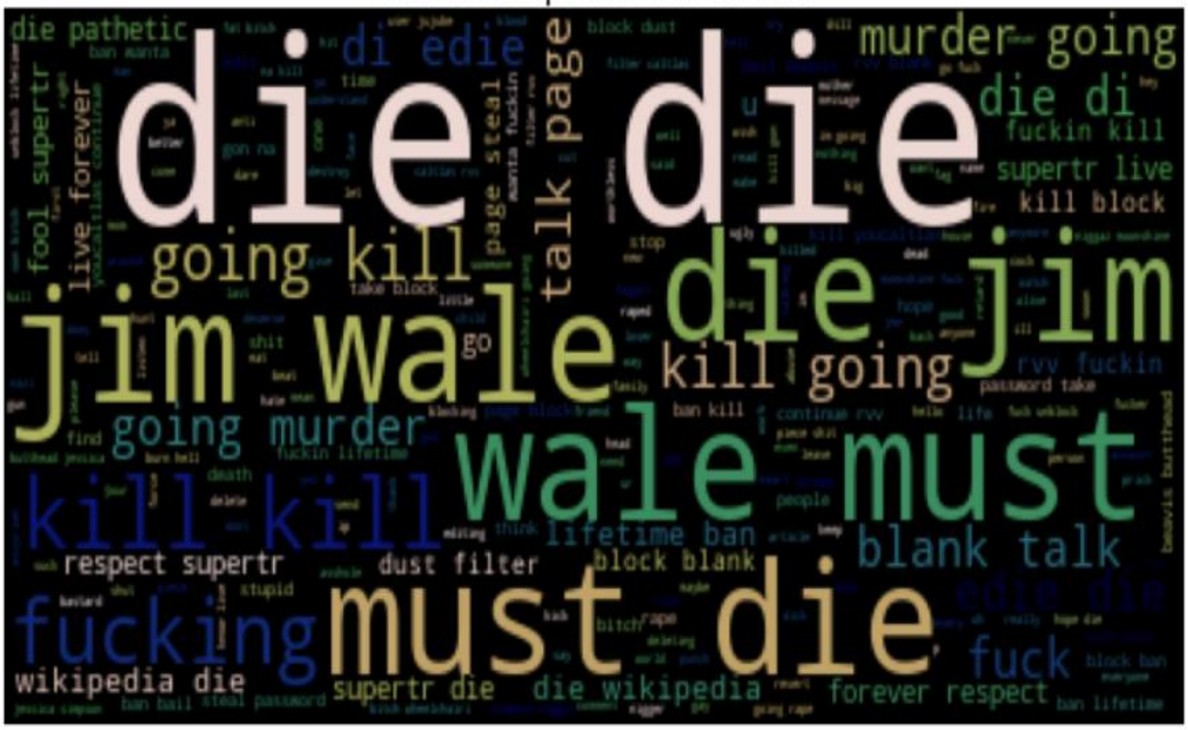
```
df_r=df.loc[:,['comment_text','rude']]
wcloud(df_r,'rude')
```

Words frequented in rude




```
df_t=df.loc[:,['comment_text','threat']]
wcloud(df_t,'threat')
```

Words frequented in threat



```
df_a=df.loc[:,['comment_text','abuse']]
wcloud(df_a,'abuse')
```

Words frequented in abuse


```
df_l=df.loc[:,['comment_text','loathe']]
wcloud(df_l,'loathe')
```

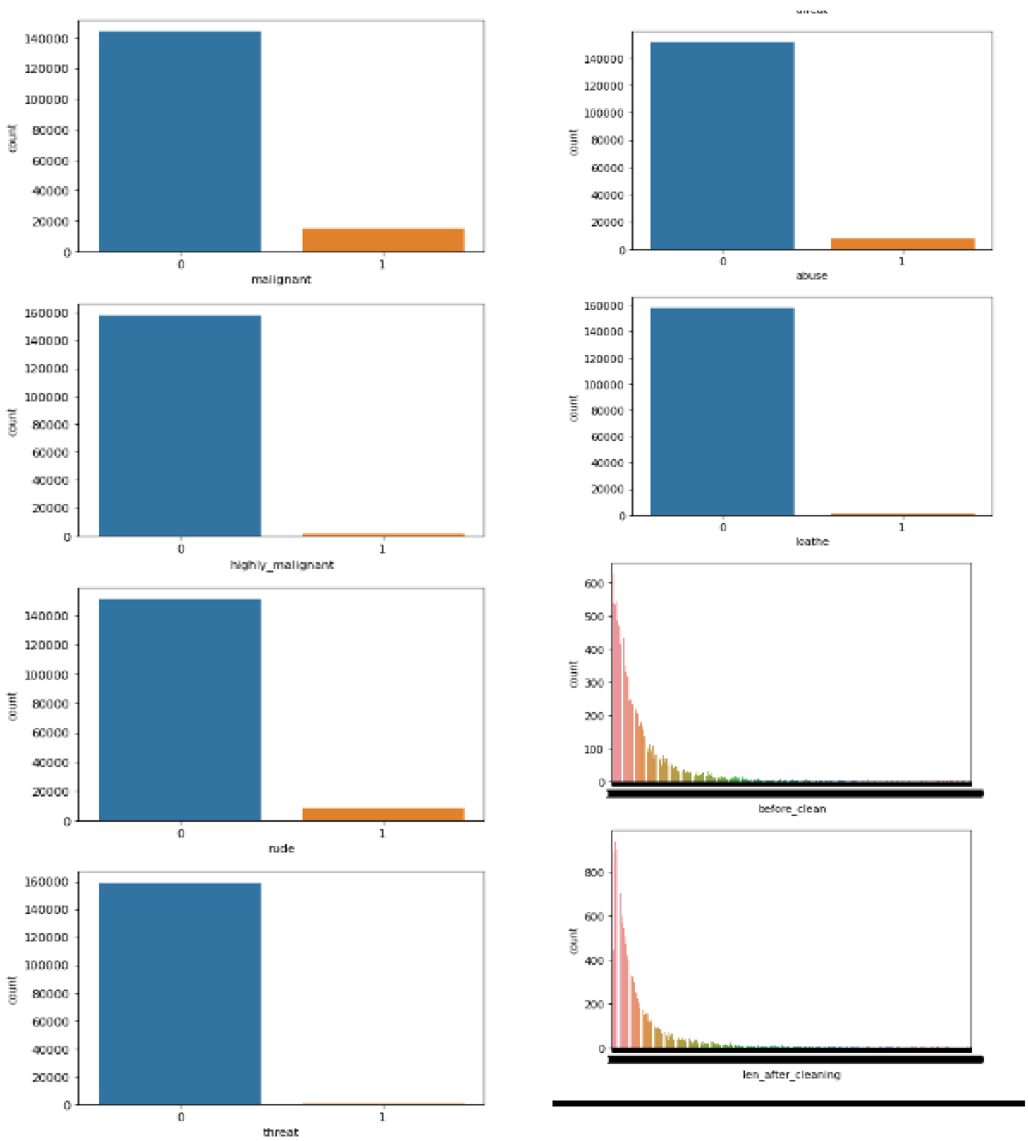
Words frequented in loathe

Plot all features using countplot:-

code:-

```
feat=df.columns[1:]
for col in feat:
    sns.countplot(df[col])
    plt.show()
```

output:-



Observation:-

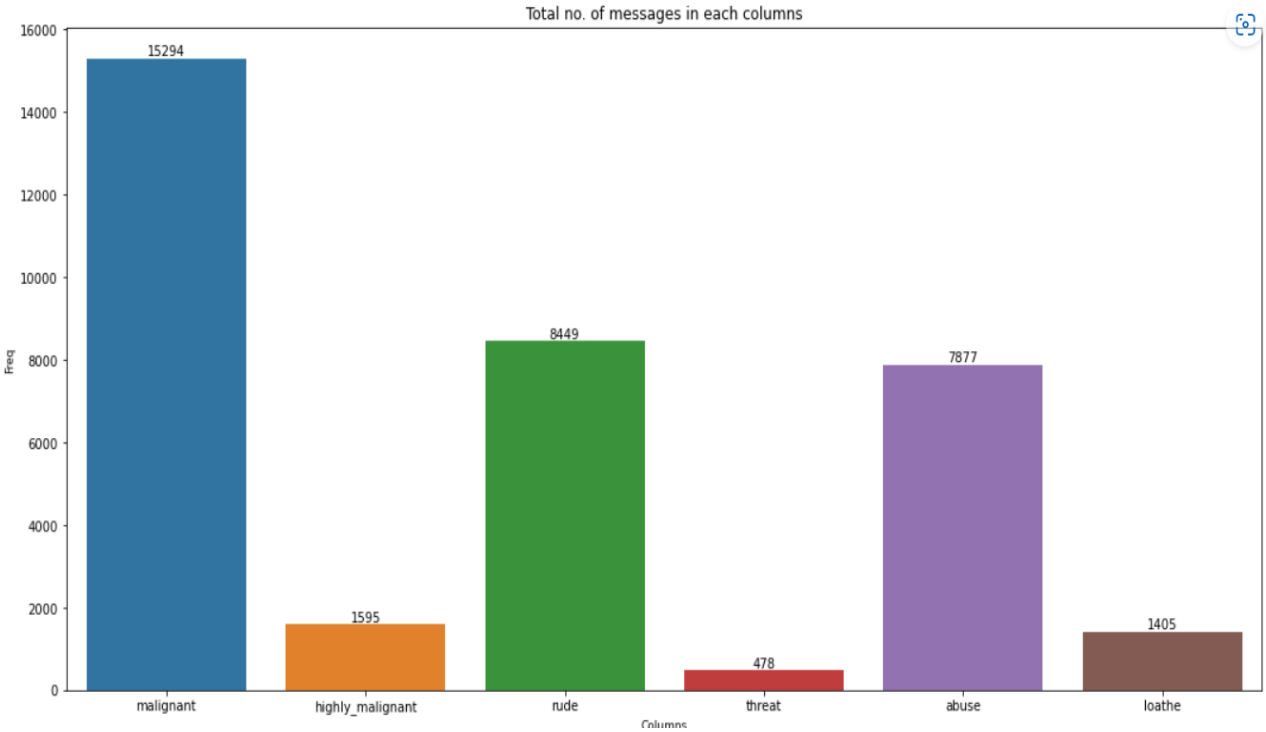
- ✓ Here in the first graph of malignant we can clearly observe that most of the messages are not malignant.
- ✓ In the second image we can clearly observe that there are very less highly malignant messages.
- ✓ Same in third picture there are few rude comments in the dataset.
- ✓ In 4th we can clearly see that there are very few cases/almost negligible of threat comments
- ✓ In 5th image we can clearly see that there are some messages with abusive language.
- ✓ While in the sixth image we can clearly see that there are very few cases of loathe messages.
- ✓ In 7th image we can see the no. of words in each rows
- ✓ In 8th image we can see the cleaned no. of remaining words in each row.

plot and visualize count of each columns:-

code:-

```
# lets plot and visualize count of each columns
plt.figure(figsize=(18,9))
ax=sns.barplot(counts.index,counts.values)
plt.title("Total no. of messages in each columns")
plt.ylabel('Freq', fontsize=9)
plt.xlabel('Columns',fontsize=9)
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()
```

Output:-



Describing Dataset:-

```
# lets check the statistical description of all the columns
df.describe()
```

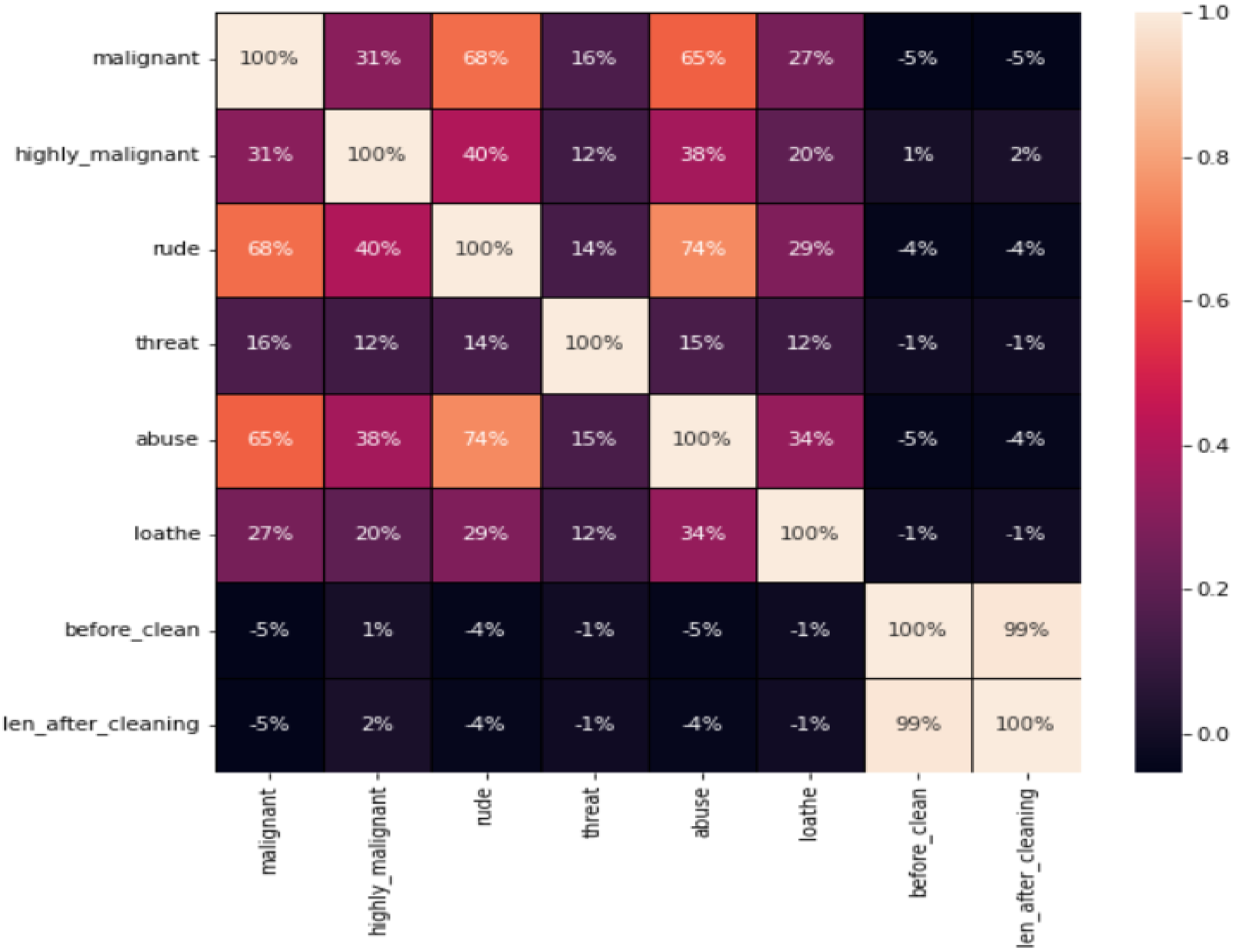
	malignant	highly_malignant	rude	threat	abuse	loathe	before_clean	len_after_cleaning
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805	394.138847	241.114238
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420	590.725381	377.602191
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	96.000000	56.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	205.000000	123.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	436.000000	263.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5000.000000	5000.000000

Observation:-

- ✓ Here we can see that only 2 values are present in all the columns i.e. 0 and 1.
- ✓ Low score of standard devaiation tells us that the data is not spreaded.
- ✓ there is difference in mean and median which tells us that some skewness is present.
- ✓ very low difference in 75% and max shows that there are no outliers present in the dataset.

Correlation Checking:-

```
plt.figure(figsize=(9,8))
sns.heatmap(df.corr(),linewidth=0.5, linecolor='black',fmt='.0%',annot=True)
plt.show()
```



Model Building:-

➢ Train_Test_Split:-

The Train_Test_Split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

```
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=.30,random_state=233)
```

```
#cheking shape of all variable
print("train_x shape =",train_x.shape)
print("test_x shape =",test_x.shape)
print("train_y shape =",train_y.shape)
print("test_y shape =",test_y.shape)
```

➢ Model Selection:-

```
#Importing required libraries
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import f1_score,precision_score, multilabel_confusion_matrix, accuracy_score,jaccard_score, recall_score, hamming_loss
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import cross_val-score
```

```
#Initializing the instance of the model
svc = LinearSVC()
lr = LogisticRegression(solver='lbfgs')
mnb = MultinomialNB()
lgb = LGBMClassifier()
sgd = SGDClassifier()
rf = RandomForestClassifier()
```

```python
def print_score(y_pred,clf):
    print('classifier:',clf.__class__.__name__)
    print("Jaccard score: {}".format(jaccard_score(y_test,y_pred,average='micro')))
    print("Accuracy score: {}".format(accuracy_score(y_test,y_pred)))
    print("f1_score: {}".format(f1_score(y_test,y_pred,average='micro')))
    print("Precision : ", precision_score(y_test,y_pred,average='micro'))
    print("Recall: {}".format(recall_score(y_test,y_pred,average='micro')))
    print("Hamming loss: ", hamming_loss(y_test,y_pred))
    print("Confusion matrix:\n ", multilabel_confusion_matrix(y_test,y_pred))
    print('========================================\n')
```

```python
#models with evaluation using OneVsRestClassifier
for classifier in [svc,lr,mnb,sgd,lgb,rf]:
   clf = OneVsRestClassifier(classifier)
   clf.fit(x_train,y_train)
   y_pred = clf.predict(x_test)
   print_score(y_pred, classifier)
```

Run and selecte Best models:-

```
classifier: LinearSVC
Jaccard score: 0.8478403520284093
Accuracy score: 0.9176554144385026
f1_score: 0.9176554144385026
Precision :  0.9176554144385026
Recall: 0.9176554144385026
Hamming loss:  0.08234458556149733
Confusion matrix:
  [[[ 2850  2018]
  [  257 42747]]

 [[45344   620]
  [ 1573   335]]

 [[46511   317]
  [  907   137]]

 [[45861   748]
  [  657   606]]

 [[47134   210]
  [  431    97]]

 [[47729    27]
  [  108     8]]

 [[47861     2]
  [    9     0]]]
========================================
```

```
classifier: LogisticRegression
Jaccard score: 0.8443875093910732
Accuracy score: 0.9156291778074866
f1_score: 0.9156291778074866
Precision :  0.9156291778074866
Recall: 0.9156291778074866
Hamming loss:  0.08437082219251336
Confusion matrix:
  [[[ 2130  2738]
  [   97 42907]]

 [[45521   443]
  [ 1685   223]]

 [[46676   152]
  [  991    53]]

 [[45991   618]
  [  661   602]]

 [[47261    83]
  [  481    47]]

 [[47751     5]
  [  115     1]]

 [[47863     0]
  [    9     0]]]
==========================================

classifier: MultinomialNB
Jaccard score: 0.8337546924078756
Accuracy score: 0.9093415775401069
f1_score: 0.9093415775401069
Precision :  0.9093415775401069
Recall: 0.9093415775401069
Hamming loss:  0.09065842245989304
Confusion matrix:
  [[[ 1271  3597]
  [   23 42981]]

 [[45799   165]
  [ 1852    56]]

 [[46823     5]
  [ 1040     4]]

 [[46038   571]
  [  774   489]]

 [[47342     2]
  [  526     2]]

 [[47756     0]
  [  116     0]]
```

```
 [[47863     0]
  [    9     0]]]
========================================

classifier: SGDClassifier
Jaccard score: 0.8368858277535829
Accuracy score: 0.9112007018716578
f1_score: 0.9112007018716578
Precision :  0.9112007018716578
Recall: 0.9112007018716578
Hamming loss:  0.08879929812834225
Confusion matrix:
  [[[ 1407  3461]
  [   10 42994]]

 [[45906    58]
  [ 1899     9]]

 [[46731    97]
  [ 1001    43]]

 [[46075   534]
  [  739   524]]

 [[47268    76]
  [  484    44]]

 [[47731    25]
  [  109     7]]

 [[47863     0]
  [    9     0]]]
========================================

classifier: LGBMClassifier
Jaccard score: 0.8471630042636931
Accuracy score: 0.9172585227272727
f1_score: 0.9172585227272727
Precision :  0.9172585227272727
Recall: 0.9172585227272727
Hamming loss:  0.08274147727272728
Confusion matrix:
  [[[ 2485  2383]
  [  161 42843]]

 [[45585   379]
  [ 1750   158]]

 [[46623   205]
  [  935   109]]

 [[45822   787]
  [  585   678]]

 [[47207   137]
  [  428   100]]
```

```
 [[47694    62]
  [   93    23]]

 [[47855     8]
  [    9     0]]]
========================================
classifier: RandomForestClassifier
Jaccard score: 0.8460589233379608
Accuracy score: 0.9166109625668449
f1_score: 0.9166109625668449
Precision :  0.9166109625668449
Recall: 0.9166109625668449
Hamming loss:  0.08338903743315508
Confusion matrix:
  [[[ 2429  2439]
  [  195 42809]]

 [[45646   318]
  [ 1746   162]]

 [[46562   266]
  [  886   158]]

 [[45775   834]
  [  599   664]]

 [[47217   127]
  [  444    84]]

 [[47748     8]
  [  113     3]]

 [[47863     0]
  [    9     0]]]
========================================
```

we get best accuracy score from **LinearSVC**

➢ ***Hyperparameter Tuning:-***

Code:-

```python
#Creating parameter list to pass in GridSearchCV
param = {
         'estimator__penalty': ['l1'],
         'estimator__loss': ['hinge','squared_hinge'],
         'estimator__multi_class': ['ovr','crammer_singer'],
         'estimator__dual': [False],
         'estimator__intercept_scaling': [2,4,5],
         'estimator__C': [2]
         }
```

```python
from sklearn.model_selection import GridSearchCV
svc = OneVsRestClassifier(LinearSVC())
GCV =  GridSearchCV(svc,param,cv = 3, verbose =0,n_jobs=-1)
GCV.fit(x_train,y_train)
```

Output:-

```
GridSearchCV(cv=3, estimator=OneVsRestClassifier(estimator=LinearSVC()),
             n_jobs=-1,
             param_grid={'estimator__C': [2], 'estimator__dual': [False],
                         'estimator__intercept_scaling': [2, 4, 5],
                         'estimator__loss': ['hinge', 'squared_hinge'],
                         'estimator__multi_class': ['ovr', 'crammer_singer'],
                         'estimator__penalty': ['l1']})
```

Checcking Best Parameter:-

```python
GCV.best_par
ams_
```

```
{'estimator__C': 2,
 'estimator__dual': False,
 'estimator__intercept_scaling': 2,
 'estimator__loss': 'hinge',
 'estimator__multi_class': 'crammer_singer',
 'estimator__penalty': 'l1'}
```

Creat Final Model:-

```
model = OneVsRestClassifier(LinearSVC(C=2,dual = False, loss='hinge',multi_class='crammer_singer', penalty ='l1',intercept_scaling=2))
model.fit(x_train,y_train)
y_pred = model.predict(x_test)

print("Jaccard score: {}".format(jaccard_score(y_test,y_pred,average='micro')))
print("Accuracy score: {}".format(accuracy_score(y_test,y_pred)))
print("f1_score: {}".format(f1_score(y_test,y_pred,average='micro')))
print("Precision : ", precision_score(y_test,y_pred,average='micro'))
print("Recall: {}".format(recall_score(y_test,y_pred,average='micro')))
print("Hamming loss: ", hamming_loss(y_test,y_pred))
print("\nConfusion matrix: \n", multilabel_confusion_matrix(y_test,y_pred))
```

```
Jaccard score: 0.8479830148619958
Accuracy score: 0.9177389705882353
f1_score: 0.9177389705882353
Precision :  0.9177389705882353
Recall: 0.9177389705882353
Hamming loss:  0.0822610294117647

Confusion matrix:
 [[[ 2889  1979]
  [  279 42725]]

 [[45465   499]
  [ 1633   275]]

 [[46575   253]
  [  904   140]]

 [[45764   845]
  [  598   665]]

 [[47064   280]
  [  411   117]]

 [[47685    71]
  [  104    12]]

 [[47852    11]
  [    9     0]]]
```

Here we have successfully improved accuracy score from 91.76 to 91.77%.

- ***SHOW PREDICTED RESULTS:-***

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

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

159571 rows × 1 columns

Saving Model:-

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

Interpretation of the Results:-

- LinearSVM and Random Forest models perform best.
- At first we get only 91.76% accuracy from LinearSVM But after parameter tuning we get 91.77% accuracy.There is no defference in accuracy score after parameter tuning.

- Using hyper parameter tunning we can improve our model accuracy, But here in this model the accuracy did not increased.

- It is always advised to all of us that atleast we need to use 5 Algorithm in order to figure out which one is performing best among them and we choose that one and we send that for hyper parameter tuning to know that best parameter .

CONCLUSION

For any of machine learning project my suggestion is first you have to understand the problem on ground level .if you don't allow yourself to work with diligence .if you don' t work harder anything that you are doing or will do , not only in case of machine learning but also in life cycle would be futile. Maybe, my endeavour assist you when ever you will get stuck

- For future improvements, following step we thought to took-
  - Replacing model with a latest/different model
  - Using other robust datasets
  - More focus on NLP properties

- It would seem that better performance might be achieved if multiple learners were combined.

Reference:-

I have also used few external resources that helped me to complete this project successfully. Below are the external resources that were used to fulfill my project.

- https://www.google.co.in/
- https://github.com/
- https://www.kaggle.com/
- https://www.youtube.com/index
- https://careerkarma.com/blog/nlp-projects/