FLIP ROBO

Malignant Comments Classifier

Submitted by:

Dipak Someshwar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Khushboo Garg.

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thank and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Dipak Someshwar

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. The loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).

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

- Import library and load the train and test dataset.

In [1]:
```python
# Import Library:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

In [2]:
```python
# Loading the train dataset:
train=pd.read_csv('Train.csv')
train.head()
```

Out[2]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

In [3]:
```python
# Loading the test  dataset:
test=pd.read_csv('Test.csv')
test.head()
```

Out[3]:

	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.

- Display train and test dateset datatypes and sum of null values.

In [8]:
```
# Here checking train data type:
train.dtypes
```

Out[8]:
```
id                  object
comment_text        object
malignant            int64
highly_malignant     int64
rude                 int64
threat               int64
abuse                int64
loathe               int64
dtype: object
```

In [9]:
```
# Here checking test data type:
test.dtypes
```

Out[9]:
```
id              object
comment_text    object
dtype: object
```

In [10]:
```
#Checking sum null values
train.isna().sum()
```

Out[10]:
```
id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
```

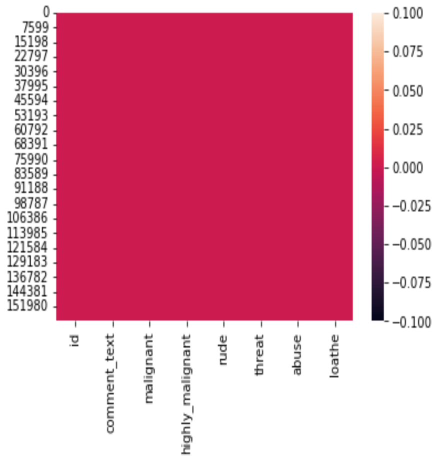
In [11]:
```
#Checking sum null values
test.isna().sum()
```

Out[11]:
```
id              0
comment_text    0
dtype: int64
```

- Display null values of columns using heatmap.

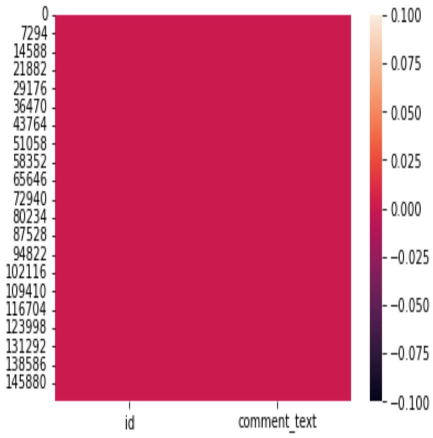
In [12]:
```
#Checking for null values using heatmap
sns.heatmap(train.isnull())
```

Out[12]: <AxesSubplot:>



In [13]:
```
#Checking for null values using heatmap
sns.heatmap(test.isnull())
```

Out[13]: <AxesSubplot:>



- Summary Statistics.

In [14]:
```
#summary statistics.
train.describe().style.background_gradient()
```

Out[14]:

	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 [15]:
```
#summary statistics.
test.describe().style.background_gradient()
```

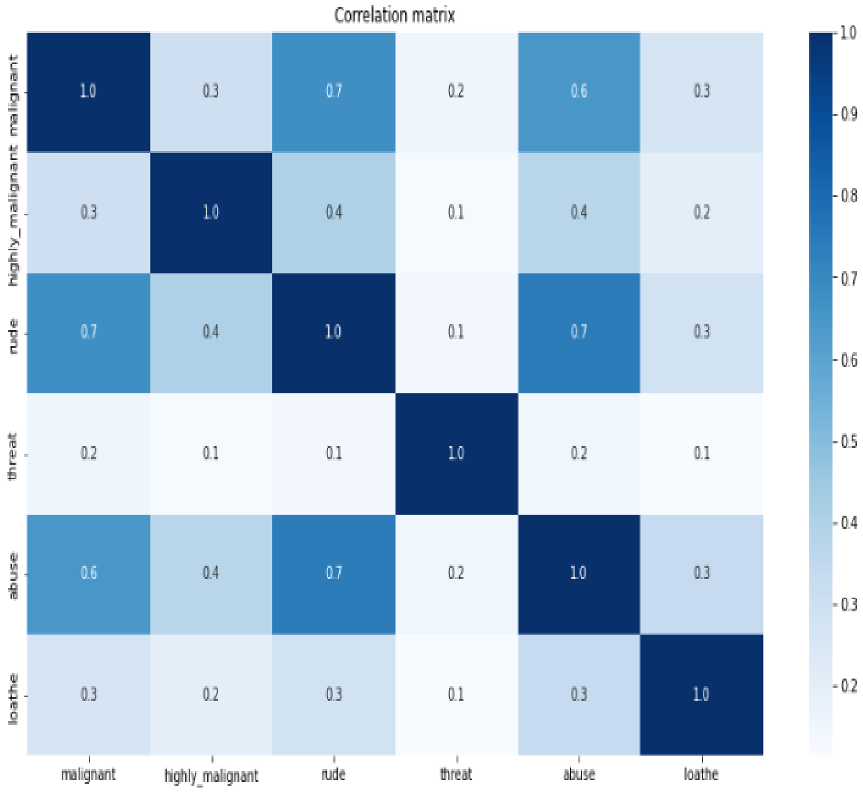
Out[15]:

	id	comment_text
count	153164	153164
unique	153164	153164
top	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll ever be whats up with you and hating you sad mofuckas...i should bitch slap ur pethedic white faces and get you to kiss my ass you guys sicken me. Ja rule is about pride in da music man. dont diss that shit on him. and nothin is wrong bein like tupac he was a brother too...fuckin white boys get things right next time.,
freq	1	1

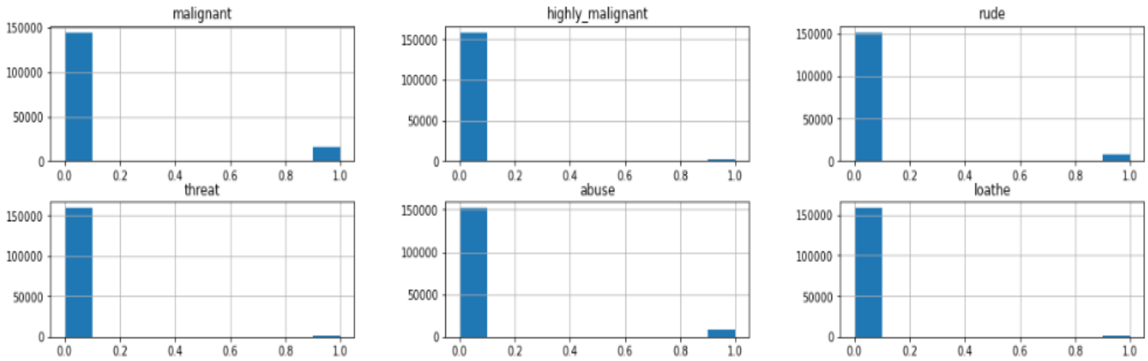
- Data Exploration.

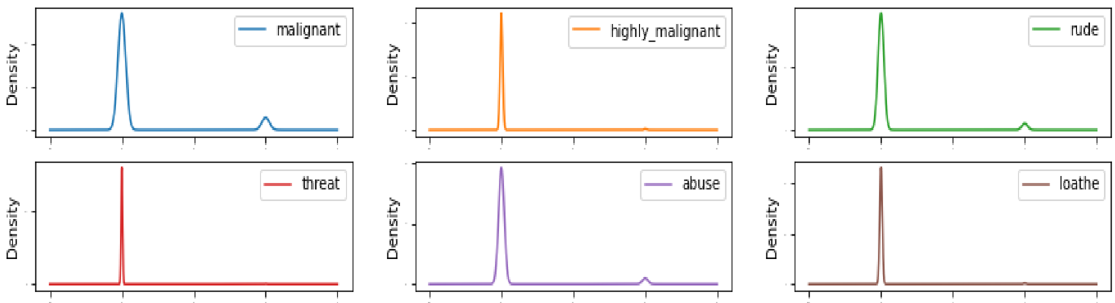
- Data Exploration.

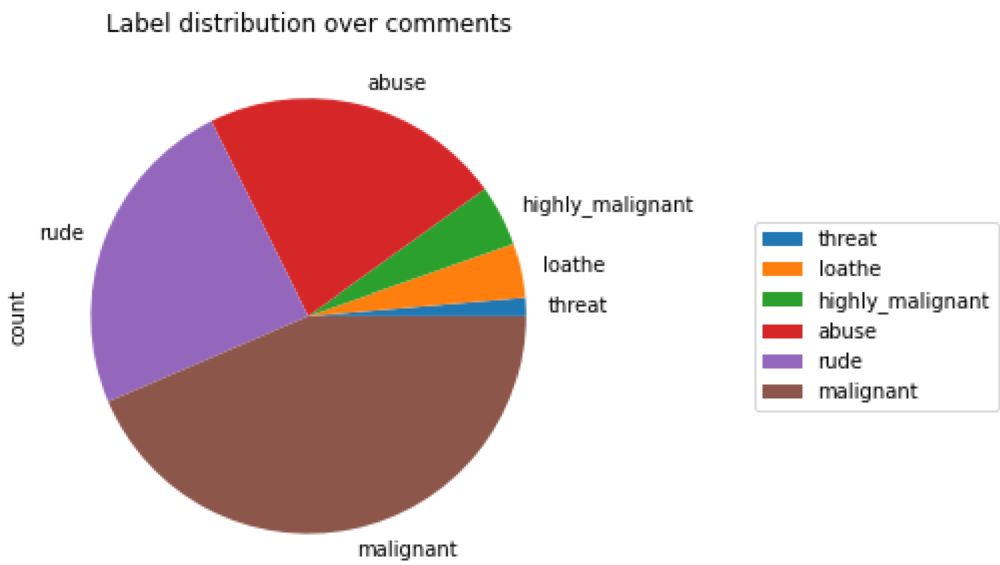
In [17]:
```python
#check correlation matrix with heatmap.
corr_mat = train.corr()
plt.figure(figsize=(15,8))
sns.heatmap(corr_mat, annot=True, fmt = '.1f',cmap = 'Blues')
plt.title('Correlation matrix')
plt.show()
```



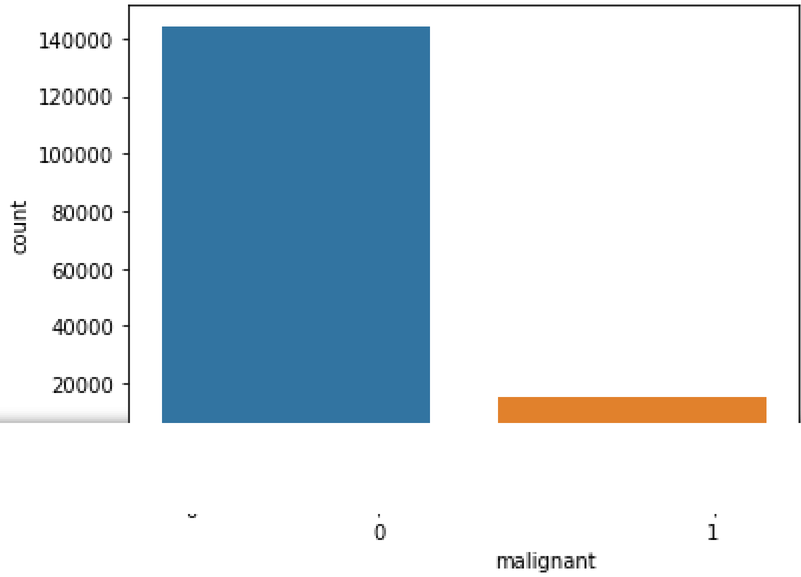
* Data Visualization.



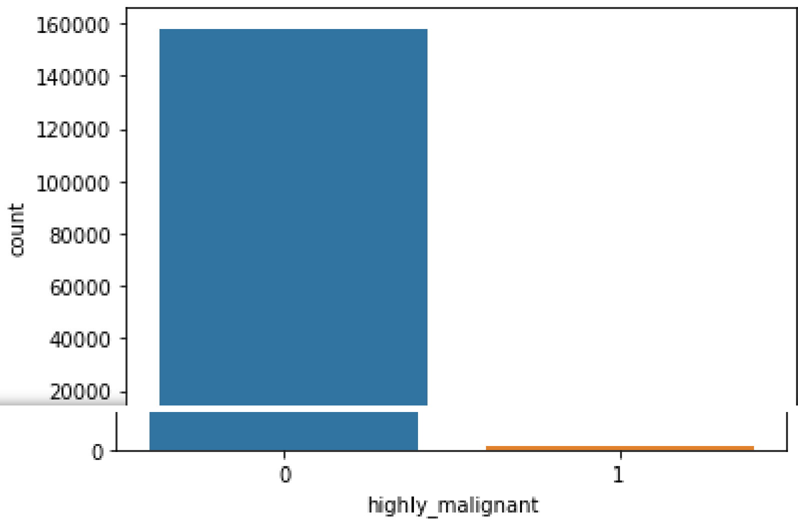




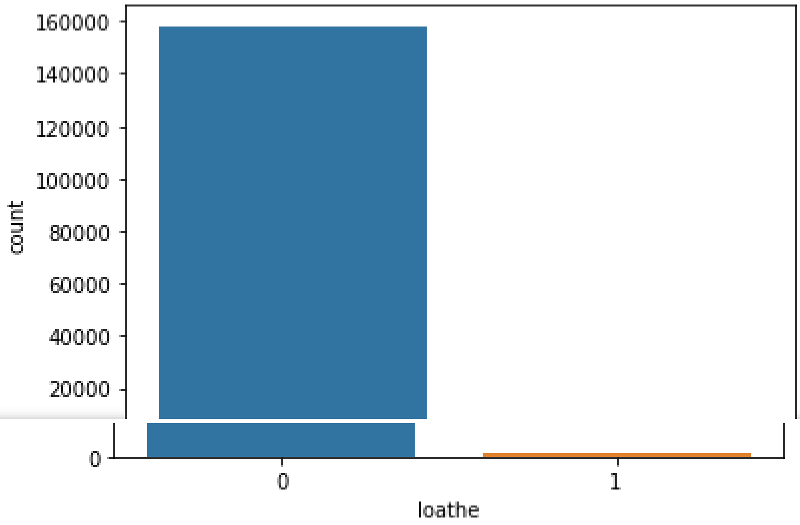
```
malignant


0    144277
1     15294
Name: malignant, dtype: int64
```

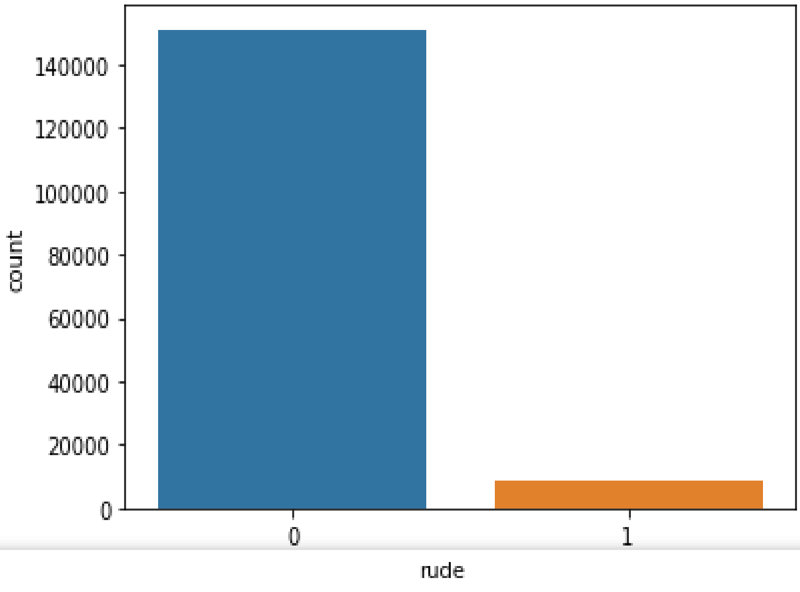


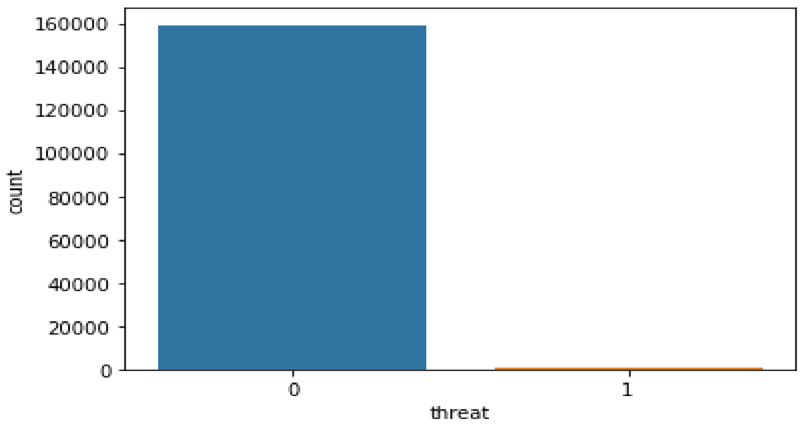

```
highly_malignant


0    157976
1      1595
Name: highly_malignant, dtype: int64
```



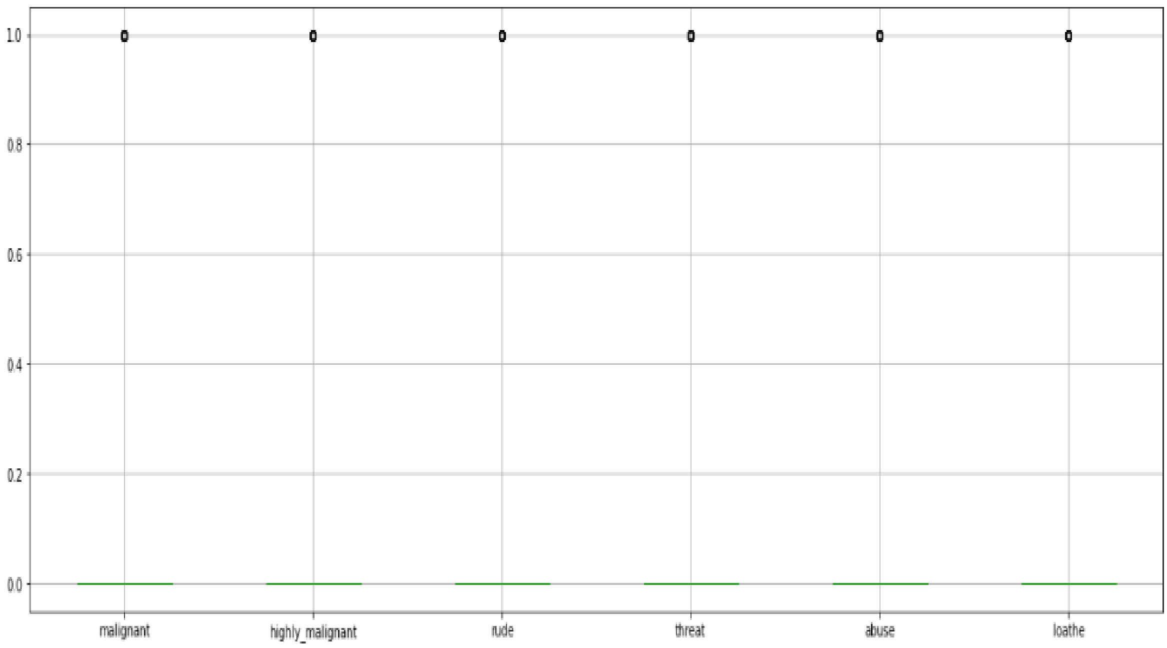
```
loathe


0    158166
1      1405
Name: loathe, dtype: int64
```







- Display boxplot of columns for outliers.



- Remove stopwords and punctuations

In [22]:
```
from nltk.stem import WordNetLemmatizer
import nltk
from nltk.corpus import  stopwords
import string
```

In [23]:
```
train['length'] = train['comment_text'].str.len()
train.head(2)
```

Out[23]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112

In [24]:
```
# Convert all messages to lower case
train['comment_text'] = train['comment_text'].str.lower()

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

# Replace URLs with 'webaddress'
train['comment_text'] = 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)
train['comment_text'] = train['comment_text'].str.replace(r'£|\$', 'dollers')

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


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


train['comment_text'] = 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') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])
train['comment_text'] = train['comment_text'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))
```

- Create text into vectors using TF-IDF and Train-Test Splitting.

In [33]:
```python
#  Convert text into vectors using TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vec = TfidfVectorizer(max_features = 10000, stop_words='english')
features = tf_vec.fit_transform(train['comment_text'])
x = features
```

In [34]:
```python
train.shape
test.shape
```

Out[34]: (153164, 2)

In [35]:
```python
y=train['bad']
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.30)
```

In [36]:
```python
y_train.shape,y_test.shape
```

Out[36]: ((111699,), (47872,))

- Run and evaluate selected models.

In [37]:
```python
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)

LG.fit(x_train, y_train)

y_pred_train = LG.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = LG.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9595520103134316
Test accuracy is 0.9552974598930482
[[42729   221]
 [ 1919  3003]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.93      0.61      0.74      4922

    accuracy                           0.96     47872
   macro avg       0.94      0.80      0.86     47872
weighted avg       0.95      0.96      0.95     47872
```

In [38]:

```python
# DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
y_pred_train = DT.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = DT.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9988898736783678
Test accuracy is 0.940194685828877
[[41627  1323]
 [ 1540  3382]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     42950
           1       0.72      0.69      0.70      4922

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

In [40]:

```python
# RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9988719684151156
Test accuracy is 0.9550050133689839
[[42404   546]
 [ 1608  3314]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.86      0.67      0.75      4922

    accuracy                           0.96     47872
   macro avg       0.91      0.83      0.86     47872
weighted avg       0.95      0.96      0.95     47872
```

In [41]:

```python
# xgboost
import xgboost

xgb = xgboost.XGBClassifier()
xgb.fit(x_train, y_train)
y_pred_train = xgb.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = xgb.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9614052050600274
Test accuracy is 0.9526236631016043
[[42689   261]
 [ 2007  2915]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     42950
           1       0.92      0.59      0.72      4922

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

In [42]:

```
# AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=100)

ada.fit(x_train, y_train)
y_pred_train = ada.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = ada.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.951118631321677
Test accuracy is 0.9490307486631016
[[42553   397]
 [ 2043  2879]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     42950
           1       0.88      0.58      0.70      4922

    accuracy                           0.95     47872
   macro avg       0.92      0.79      0.84     47872
weighted avg       0.95      0.95      0.94     47872
```

In [43]:

```
# KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
y_pred_train = knn.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = knn.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.922300110117369
Test accuracy is 0.9173629679144385
[[42809   141]
 [ 3815  1107]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     42950
           1       0.89      0.22      0.36      4922

    accuracy                           0.92     47872
   macro avg       0.90      0.61      0.66     47872
weighted avg       0.91      0.92      0.89     47872
```

In [44]:

```
# RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
cvs=cross_val_score(RF, x, y, cv=10, scoring='accuracy').mean()
print('cross validation score :',cvs*100)
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9988630157834896
Test accuracy is 0.9554645721925134
cross validation score : 95.65522481410615
[[42402   548]
 [ 1584  3338]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.86      0.68      0.76      4922

    accuracy                           0.96     47872
   macro avg       0.91      0.83      0.87     47872
weighted avg       0.95      0.96      0.95     47872
```

- Hypertuning of the Model.

In [63]:

```python
from sklearn.model_selection import RandomizedSearchCV
```

In [64]:

```python
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [65]:

```python
parameters = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
```

In [67]:

```python
rf_random = RandomizedSearchCV(estimator = RF, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

Hypertuning of the model:

In [63]:

```python
from sklearn.model_selection import RandomizedSearchCV
```

In [64]:

```python
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [65]:

```python
parameters = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
```

In [67]:

```python
rf_random = RandomizedSearchCV(estimator = RF, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

```
Out[55]: RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=56, scoring='neg_mean_squared_error',
                   verbose=2)
```

```
In [56]: rf_random.best_params_
```

```
Out[56]: {'n_estimators': 100,
          'min_samples_split': 5,
          'min_samples_leaf': 5,
          'max_features': 'auto',
          'max_depth': 30}
```

```
In [61]: rfc = RandomForestClassifier(n_estimators=100, min_samples_split=5, min_samples_leaf=5, max_features='auto', max_depth=30)
rfc.fit(x_train,y_train)

y_pred_train = rfc.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))

y_pred_test = rfc.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))

rfscore = cross_val_score(rfc,x,y,cv=10)
rfc = rfscore.mean()
print("Cross Val Score is:",rfc*100)
```

```
Training accuracy is 0.9114763784814546
Test accuracy is 0.9085060160427807
Cross Val Score is: 90.95387035544353
```

- Hardware and Software Requirements and Tools Used

➢ **Language :-** Python

➢ **Tool:-** Jupyter

➢ **OS:-** Windows 10

➢ **RAM:-** 8gb

CONCLUSION:

- This Kernel investigates different models for car price prediction.
- Different types of Machine Learning methods including LinearRegression, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor and DecisionTreeRegressor in machine learning are compared and analysed for optimal solutions.
- Even though all of those methods achieved desirable results, different models have their own pros and cons.
- The RandomForestRegressor is probably the best one and has been selected for this problem.

- Finally, the RandomForestRegressor is the best choice when parameterization is the top priority.