MALIGNANT-COMMENT CLASSIFICATION

Submitted by:

Paras Malhotra

ACKNOWLEDGMENT

It gives me a great pleasure in presenting the project report on Malignant Comments Classification using Machine Learning.

Sources:

Google.com

Wikipedia

INTRODUCTION

The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions.

Problem Statement:

The expansion of social media enables users to express their opinions widely online. At the same time, this has resulted in the emergence of conflict and hate, making online environments inconvenient 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. There has been a remarkable increase in cases of cyberbullying and trolls on various social media platforms. Many celebrities and influencers 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.

Problem Background:

The background for the problem originates from the multitude of online forums, where-in people participate actively and make comments. As the comments sometimes may be abusive, insulting or even hate-based. The task was thus to build a model which could make prediction to classify the comments into various category.

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

Literature:

Literature survey is the most important in any kind of research. Most of the literature study is based on articles from websites. This literature study

attempts to construct a model based on classification techniques and how it can be applied to predict malignant comments.

The literature study gives an overview of the articles that are related to this study, the feature engineering methods and the evaluation metrics that are used to measure the performance of the algorithms.

This paper presents how to make optimal use of Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Kneighbors Classifier, Ada Boost Classifier, Gradient Boosting Classifier and XGBoost Classifier. The system will give effective result on knowing the causes for hatred and offensive comments.

Motivation:

Comment classification regarding toxicity has been intensively researched in the past few years, largely in the context of social media data where researches have applied various machine learning techniques to try and tackle the problem of toxicity related to comments.

ANALYTICAL PROBLEM FRAMING

Mathematical/Analytical Modelling:

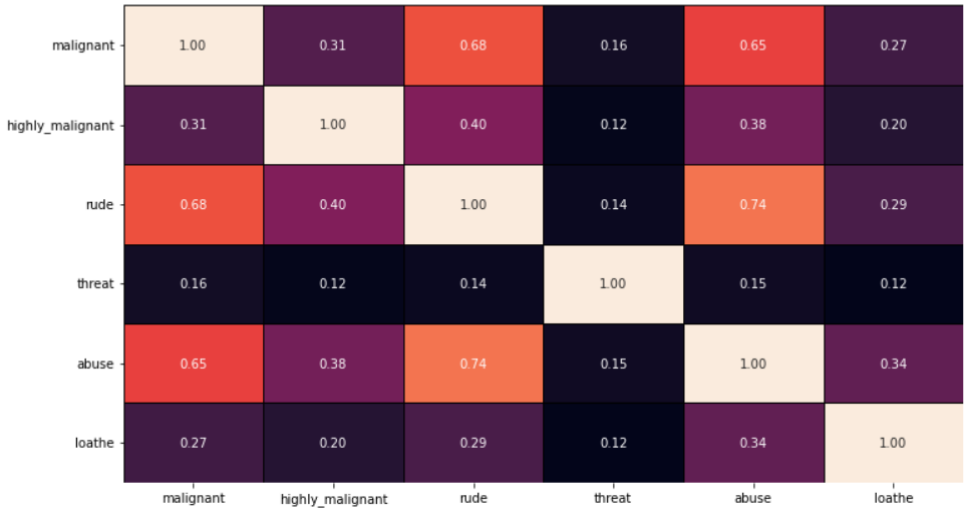
This experiment is done to pre-process the data and evaluate the prediction accuracy of the models. The experiment has multiple stages that are required to get the prediction results. These stages can be defined as:

- Pre-Processing: The dataset will be checked and pre-processed using certain methods. These methods have various ways of handling data. Thus, the pre-processing is done on multiple iterations where each time the accuracy will be evaluated with the used combination.
- Data splitting: Dividing the dataset into two parts to train the model with one and use the other in the testing. The dataset will be split 75% for training and 25% for testing.
- Evaluation: The accuracy of both datasets will be evaluated by measuring the accuracy score, confusion matrix and classification report when training the model with an evaluation of the on the test dataset with that are being predicted by the model.
- Correlation: Correlation analysis defines the strength of a relationship between two variables, which can be between two independent

variables or one independent and one dependent variable. Correlation between the available features and output will be evaluated to identify whether the features have a negative, positive or zero correlation with the output variable.

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

```
<AxesSubplot:>
```

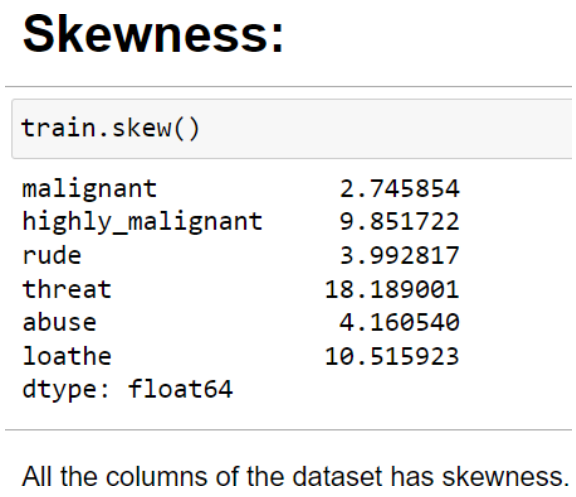


- Describe of the dataset: Describe of the dataset is plotted using the heatmap which shows us the mean, standard deviation, maximum and minimum value of each column in the given dataset.

Describe of dataset using heatmap:

```
plt.figure(figsize=(15,12))
sns.heatmap(round(train.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt="f")
plt.xticks(fontsize=18)
plt.yticks(fontsize=12)
plt.title("Variable Summary")
plt.show()
```

- Evaluation Metrics: The prediction accuracy will be evaluated by measuring the accuracy score, confusion matrix and classification report. Accuracy score is the ratio of number of correct predictions to the total number of input samples. It works well only if there are equal number of samples belonging to each class. A confusion matrix is an NxN matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 score and support of our trained classification model.
- Skewness: Skewness is a measure of the asymmetry of the probability distribution of a real valued random variable about its mean.



Data Description:

In Machine Learning, the training data set is the actual dataset used to train the model for predicting the toxicity in the comments.

The dataset contains the training set, which has 1,59,571 samples and the test dataset which contains 1,53,164 samples. All the data samples contain 8 fields which includes ‘id’, ‘comment_text’, ‘malignant’, ‘highly_malignant’, ‘rude’, ‘threat’, ‘abuse’, ‘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 dataset includes:

- ID: It includes unique id associated with each comment text given.
- Comment text: This column contains the comments extracted from various social media platforms.
- Malignant: It is the label column, which includes values 0 and 1, denoting if the column 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.

Data Pre-Processing:

Data pre-processing is an important step in machine learning to get highly accurate and reliable result. Data pre-processing helps in increasing the quality of data by filling in missing data's (NaN values), removing outliers, scaling the data.

There are many steps involved in data pre-processing:

- Data Cleaning helps to impute the missed values and removing outliers from the dataset.
- Data Integration integrates data from multiple sources into single dataset.
- Data Transformation such as normalization helps in improving the accuracy and efficiency of algorithms involved in machine learning.
- Data Reduction reduces the data size by dropping out redundant features using feature selection and feature extraction techniques.

Treating null values

Sometimes there can be certain columns which may contain the null values used to indicate the missing oí unknown values. In our dataset there areno null values present.

Exploratory Data Analysis:

During the exploratory Data Analysis, I found that many attributes of comments outside of the words themselves may be useful in predicting whether they are toxic. The feature I added is comment length in characters.

Cleaning:

There are IP addresses and punctuations present in some comments. This data may compromise the model's ability while providing accuracy. Hence, I have used Natural Language Processing techniques to strip all IP addresses, punctuations, replacing 10 digits phone number etc. from comments and applied lemmatization to return the base form of words and then added a new feature clean_length, which gives the length of the comments after applying NLP techniques.

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

#Replace email address 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 sumbols with 'moneysymb'
train['comment_text'] = train['comment_text'].str.replace(r'£|\$', 'dollars')

#Replace 10 digit phone numbers with 'phonenumber'
train['comment_text'] = train['comment_text'].str.replace(r'\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$', 'phonenumber')

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

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

stop_words = set(stopwords.words('english') + ['u','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))

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

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

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length	clean_length
0	0000997932d777bf	explanationwhytheeditsmadeundermyusernamehardc...	0	0	0	0	0	0	264	217
1	000103f0d9cfb60f	d'awwlhematchesthisbackgroundcolouri'mseemingl...	0	0	0	0	0	0	112	97
2	000113f07ec002fd	heyman,i'mreallynottryingtoeditwar.it'sjusttha...	0	0	0	0	0	0	233	192
3	0001b41b1c6bb37e	moreican'tmakeanyrealsuggestionsonimprovementi...	0	0	0	0	0	0	622	503
4	0001d958c54c6e35	you,sir,aremyhero.anychanceyourememberwhatpage...	0	0	0	0	0	0	67	55
...	...	...	...	...	...	...	...	...	...	...
159566	ffe987279560d7ff	":::::andforthesecondtimeofasking,whenyourview...	0	0	0	0	0	0	295	243
159567	ffea4adeee384e90	youshouldbeashamedofyourselfthatisahorriblethi...	0	0	0	0	0	0	99	74
159568	ffee36eab5c267c9	spitzerumm,theresnoactualarticleforprostitutio...	0	0	0	0	0	0	81	66
159569	fff125370e4aaaf3	anditlookslikeitwasactuallyyouwhoputonthespeed...	0	0	0	0	0	0	116	92
159570	fff46fc426af1f9a	and...ireallydon'tthinkyouunderstand.icamehere...	0	0	0	0	0	0	189	148

Vectorization:

It is necessary to vectorize text before giving it as an input to machine learning models. This is a process of converting data into numerical data that the computer can better understand. Vectorized data is usually sparse, with an

array where the features contain either counts or another way of representing the occurrence of characters or words in a string. This is done by a vector object, which stores a dictionary of characters or words and their associated integer representation, along with relevant statistics.

Here, I used tf-idf (term frequency-inverse document frequency) vectorization method. The number of features and presence of character n-grams is a parameter to tune for model optimization.

```
#converting text into vectors using TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer
tfidf= TfidfVectorizer(max_features=10000, stop_words='english')
feature= tfidf.fit_transform(train['comment_text'])
```

Dividing dataset into feature and target:

The dataset is divided into feature and target before applying machine learning models and the target data is balanced using SMOTE.

```
x=feature
y=train['bad']
```

```
y.value_counts()
```

```
0    143346
1     16225
Name: bad, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
xtrain,ytrain=sm.fit_resample(x,y)
```

```
ytrain.value_counts()
```

```
0    143346
1    143346
Name: bad, dtype: int64
```

Input-Output Relationship:

The dependent variable or target or output depends on the features or input variables given in the dataset.

Tools Used:

Hardware: The needed time to train the model depends on the capability of the used system during the experiment. Some libraries use GPU resources over the CPU to take a shorter time to train a model.

Operating System	Windows 10
Processor	CORE i3
RAM	16GB

Language: Python

- Python is widely used in numeric and scientific computing.
- Scipy is a collection of packages for mathematics, science and engineering.
- Pandas is a data analysis and modelling library.

Libraries:

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Scikit Learn

MODEL DEVELOPMENT AND EVALUATION

Model:

Classification model is used. Classification refers to a predictive modeling problem where a class label is predicted for a given set of input data. It is a supervised technique. Regression models a target prediction based on independent variables.

Logistic Regression:

Linear regression is a

learning algorithm based on supervised learning. Logistic regression is a classification algorithm, used when the value of the target variable is

categorical in nature. It is most commonly used when the data has binary input, so when it belongs to one class or the other, or is either a 0 or 1.

Logistic Regression:

```
lg=LogisticRegression()

train_x,test_x,train_y,test_y=train_test_split(xtrain,ytrain,test_size=0.20,random_state=45)
lg.fit(train_x,train_y)
pred_train=lg.predict(train_x)
pred_test=lg.predict(test_x)

print("Training Accuracy:",accuracy_score(train_y,pred_train)*100)
print("Testing Accuracy:",accuracy_score(test_y,pred_test)*100)
```

```
Training Accuracy: 75.04632596913928
Testing Accuracy: 74.33858281448926
```

Cross Validation for logistic regression:

```
from sklearn.model_selection import cross_val_score

lg.fit(train_x,train_y)
lg.score(train_x,train_y)
pred_lg = lg.predict(test_x)

lss = accuracy_score(test_y,pred_lg)
for j in range(2,10):
    lsscore = cross_val_score(lg,xtrain,ytrain,cv=j)
    ls_cv = lsscore.mean()
    print("At cv:-",j)
    print("Cross validation score is:-",ls_cv*100 )
    print("Accuracy score is :-",lss*100)
    print("\n")
```

```
At cv:- 2
Cross validation score is:- 73.79417632860353
Accuracy score is :- 74.33858281448926
```

Decision Tree Classifier:

It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a decision tree, there are two nodes, which are the decision node and leaf node. Decision nodes are used to make any decision and have multiple branches, whereas leaf nodes are the output of those decisions and do not contain any further branches.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like-structure.

1.Decision Tree Classifier:

```
parameters = {'criterion':['gini', 'entropy'],'splitter':['best','random']}
dtc = DecisionTreeClassifier()
clf = GridSearchCV(dtc,parameters)
clf.fit(train_x,train_y)

print(clf.best_params_)
```

```
{'criterion': 'gini', 'splitter': 'best'}
```

```
dtc = DecisionTreeClassifier(criterion='gini', splitter='best')
dtc.fit(train_x,train_y)
dtc.score(train_x,train_y)
pred_dtc = dtc.predict(test_x)

print("Accuracy Score:",accuracy_score(test_y,pred_dtc)*100)
print("Classification report:",classification_report(test_y,pred_dtc)*100)
print("Confusion Matrix:",confusion_matrix(test_y,pred_dtc)*100)

dtc_score = cross_val_score(dtc,xtrain,ytrain,cv=9)
dtc_cc = dtc_score.mean()
print('Cross Val Score:',dtc_cc*100)
```

```
Accuracy Score: 77.23538952545388
Classification report:               precision    recall  f1-score   support

           0       0.91      0.61      0.73     28576
           1       0.71      0.94      0.81     28763

    accuracy                           0.77     57339
   macro avg       0.81      0.77      0.77     57339
weighted avg       0.81      0.77      0.77     57339
              precision    recall  f1-score   support

           0       0.91      0.61      0.73     28576
           1       0.71      0.94      0.81     28763

    accuracy                           0.77     57339
   macro avg       0.81      0.77      0.77     57339
weighted avg       0.81      0.77      0.77     57339
              precision    recall  f1-score   support
```

KNeighbors Classifier:

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve classification problems. It's easy to implement and understand, but has a major drawback of becoming significally slow as the size of the data in use grows. It stores all available cases and classifies new cases based on a similarity measure. KNN has been used in statistical estimation and pattern recognition as a non-parametric technique.

Random Forest Classifier:

It is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness when building each individual tree and try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

1.Random Forest Classifier:

```
parameters = {'criterion':['gini', 'entropy'],'n_estimators':[100]}
rfc = RandomForestClassifier()
clf = GridSearchCV(rfc,parameters)
clf.fit(train_x,train_y)

print(clf.best_params_)
```

```
{'criterion': 'entropy', 'n_estimators': 100}
```

```
rfc = RandomForestClassifier(criterion='entropy', n_estimators=100)
rfc.fit(train_x,train_y)
rfc.score(train_x,train_y)
pred_rfc = rfc.predict(test_x)

print("Accuracy Score:",accuracy_score(test_y,pred_rfc)*100)
print("Classification report:",classification_report(test_y,pred_rfc)*100)
print("Confusion Matrix:",confusion_matrix(test_y,pred_rfc)*100)

rfc_score = cross_val_score(rfc,xtrain,ytrain,cv=9)
rfc_cc = rfc_score.mean()
print('Cross Val Score:',rfc_cc*100)
```

```
Accuracy Score: 77.93822703569995
Classification report:               precision    recall  f1-score   support

           0       0.93      0.61      0.73     28576
           1       0.71      0.95      0.81     28763

    accuracy                           0.78     57339
   macro avg       0.82      0.78      0.77     57339
weighted avg       0.82      0.78      0.77     57339
              precision    recall  f1-score   support

           0       0.93      0.61      0.73     28576
           1       0.71      0.95      0.81     28763

    accuracy                           0.78     57339
   macro avg       0.82      0.78      0.77     57339
weighted avg       0.82      0.78      0.77     57339
```

Ada Boost Classifier:

An Ada boost or Adaptive boosting is one of the ensemble boosting classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

2.Ada Boost Classifier:

```
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1.0, algorithm='SAMME')
abc.fit(train_x,train_y)
abc.score(train_x,train_y)
pred_abc = abc.predict(test_x)

print("Accuracy Score:",accuracy_score(test_y,pred_abc)*100)
print("Classification report:",classification_report(test_y,pred_abc)*100)
print("Confusion Matrix:",confusion_matrix(test_y,pred_abc)*100)

abc_score = cross_val_score(abc,xtrain,ytrain,cv=9)
abc_cc = abc_score.mean()
print('Cross Val Score:',abc_cc*100)
```

```
Accuracy Score: 56.642076073876424
Classification report:               precision    recall  f1-score   support

           0       0.84      0.16      0.27     28576
           1       0.54      0.97      0.69     28763

    accuracy                           0.57     57339
   macro avg       0.69      0.57      0.48     57339
weighted avg       0.69      0.57      0.48     57339
              precision    recall  f1-score   support

           0       0.84      0.16      0.27     28576
           1       0.54      0.97      0.69     28763

    accuracy                           0.57     57339
   macro avg       0.69      0.57      0.48     57339
weighted avg       0.69      0.57      0.48     57339
              precision    recall  f1-score   support
```

Gradient Boosting Classifier:

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

3.Gradient Boosting Classifier:

```
gbc = GradientBoostingClassifier(criterion='mse', n_estimators=100, learning_rate=0.1, loss='deviance')
gbc.fit(train_x,train_y)
gbc.score(train_x,train_y)
pred_gbc = gbc.predict(test_x)

print("Accuracy Score:",accuracy_score(test_y,pred_gbc)*100)
print("Classification report:",classification_report(test_y,pred_gbc)*100)
print("Confusion Matrix:",confusion_matrix(test_y,pred_gbc)*100)

gbc_score = cross_val_score(gbc,xtrain,ytrain,cv=9)
gbc_cc = gbc_score.mean()
print('Cross Val Score:',gbc_cc*100)
```

```
Accuracy Score: 63.346064633146725
Classification report:              precision    recall  f1-score   support

           0       0.79      0.36      0.49     28576
           1       0.59      0.91      0.71     28763

    accuracy                           0.63     57339
   macro avg       0.69      0.63      0.60     57339
weighted avg       0.69      0.63      0.60     57339
              precision    recall  f1-score   support

           0       0.79      0.36      0.49     28576
           1       0.59      0.91      0.71     28763

    accuracy                           0.63     57339
   macro avg       0.69      0.63      0.60     57339
weighted avg       0.69      0.63      0.60     57339
              precision    recall  f1-score   support
```

XGBoost Classifier:

XGBoost (Extreme Gradient Boosting) belongs to family of boosting algorithms and uses the gradient boosting framework at its core. It is an optimized gradient boosting library.

XGBOOST:

```
#!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.5.1-py3-none-win_amd64.whl (106.6 MB)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.20.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.5.1
```

```
import xgboost

xgb = xgboost.XGBClassifier()
xgb.fit(train_x,train_y)
xgb.score(train_x,train_y)
pred_xgb = xgb.predict(test_x)

print("Accuracy Score:",accuracy_score(test_y,pred_xgb)*100)
print("Classification report:",classification_report(test_y,pred_xgb)*100)
print("Confusion Matrix:",confusion_matrix(test_y,pred_xgb)*100)

xgb_score = cross_val_score(xgb,xtrain,ytrain,cv=9)
xgb_cc = gbc_score.mean()
print('Cross Val Score:',xgb_cc*100)
```

```
[18:40:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicit
ly set eval_metric if you'd like to restore the old behavior.
Accuracy Score: 67.83515582762169
Classification report:              precision    recall  f1-score   support

           0       0.84      0.44      0.58     28576
           1       0.62      0.91      0.74     28763

    accuracy                           0.68     57339
   macro avg       0.73      0.68      0.66     57339
weighted avg       0.73      0.68      0.66     57339
              precision    recall  f1-score   support

           0       0.84      0.44      0.58     28576
           1       0.62      0.91      0.74     28763

    accuracy                           0.68     57339
   macro avg       0.73      0.68      0.66     57339
```

Algorithms Used:

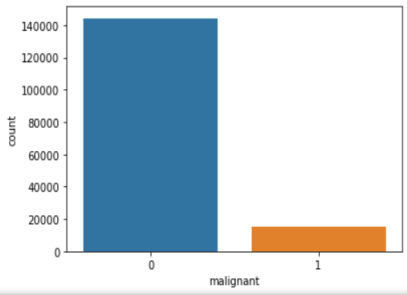
- Logistic Regression
- Decision Tree Classifier
- KNeighbors Classifier
- Random Forest Classifier
- Ada Boost Classifier
- Gradient Boosting Classifier
- XGBoost Classifier

Visualizations:

```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

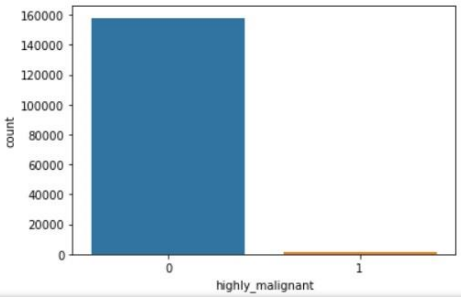
```
0    144277
1     15294
Name: malignant, dtype: int64
```




```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

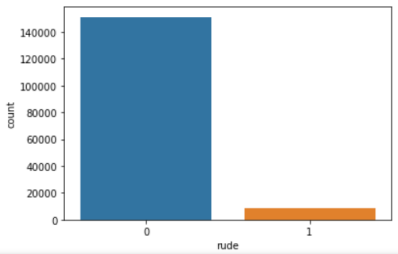
```
0    157976
1      1595
Name: highly_malignant, dtype: int64
```




```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

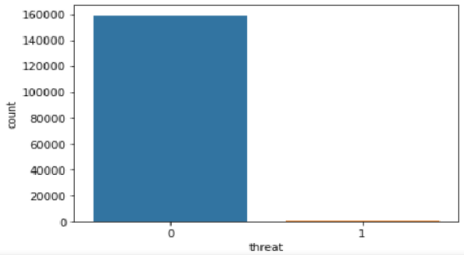
```
0    151122
1      8449
Name: rude, dtype: int64
```




```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

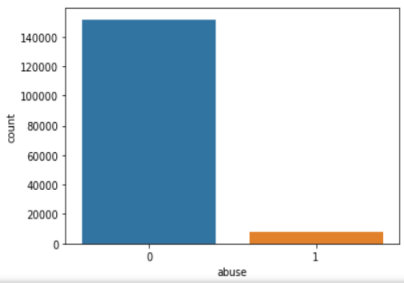
```
0    159093
1       478
Name: threat, dtype: int64
```



```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

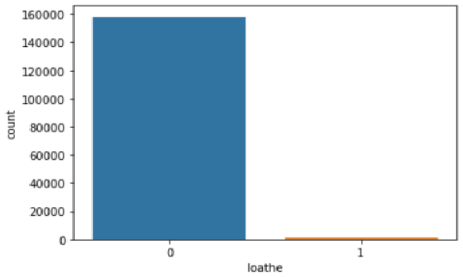
```
0    151694
1      7877
Name: abuse, dtype: int64
```




```
col=['malignant','highly_malignant','rude','threat','abuse','loathe']

for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

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



Interpretation of Results:

Many machine learning algorithms are used to predict. Using these algorithms is beneficial so that the result can be as near to the claimed results. However, the prediction accuracy of these algorithms depends heavily on the given data when training the model. If the data is in bad shape, the model will be overfitted and inefficient, which means that data pre-processing is an important part of this experiment and will affect the final results. Thus, multiple combinations of pre-processing methods need to be tested before getting the data ready to be used in training.

From Visualizations it is clear that how each feature from the dataset is relatable to the comments toxicity classification. In pre-processing, unnecessary feature is removed, skewness from the input variables are removed and web addresses, email, punctuations and phone numbers are replaced. Vectorization is done to convert the given feature into machine readable form. The data's are scaled and transformed for better training purpose. However, the chosen algorithm is Random Forest Classifier with 78% accuracy.

CONCLUSION

The objective of this research was to introduce relational learning method and to determine if it was better in default prediction over traditional prediction method. This research predicted the model with 78% accuracy and the chosen algorithm is Random Forest Classifier.

Limitations:

This study did not cover all the classification algorithms; instead, it is focused on the chosen algorithm, starting from the basic regression techniques to the advanced techniques.