MALIGNANT COMMENTS CLASSIFIER

Submitted by:

Neha Kamath

ACKNOWLEDGMENT

I would like to take this opportunity to thank my mentors at FlipRobo Technologies for their guidance and support in the completion of this project.

Index

Table of Contents

INTRODUCTION

- Background

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 un-offensive, 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 Review

- ## Analytical Modeling

For the purpose of malignant comment classifier analysis, the most important factors under consideration would be the comment text and the comment classification labels such as Malignant, Highly_malignant, Rude, Threat, Abuse, Loathe.

In the course of detailed analysis, we have applied various techniques such as Data Cleaning, Text Processing, Sentiment Analysis, Exploratory Data Analysis, Model Building, Evaluation and Selection.

- ## Data Sources

Our primary source of data for this project has been the data collected from comments posted online. The training dataset includes a total of 8 features and the total number of records is approx 1,59,000. Our test set contains 1,53,000 samples.

Here is a glimpse of our dataset:

	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

- ## Data Preprocessing Done

The data pre-processing for this particular dataset required feature engineering, checking missing values and imputing them if required, processing of text data in comments field- which included removing punctuations & special characters, splitting comments into individual words, removing stop words, stemming, lemmatization, applying count vectorizer.

Removing Punctuations: The string library contains punctuation characters. This is imported and all numbers are appended to this string. Also, we can notice that our comment_text field contains strings such as won't, didn't, etc which contain apostrophe character('). To prevent these words from being converted to wont/didnt, the character '

represented as ' in escape sequence notation is replaced by empty character in the punctuation string.

Stop Words: **Stop words** are those words that are frequently used in both written and verbal communication and thereby do not have either a positive/negative impact on our statement.E.g. is, this, us,etc. Single letter words if existing or created due to any preprocessing step do not convey any useful meaning and hence can be directly removed. Hence letters from b to z, will be added to the list of stop words imported directly.

Stemming and Lemmatizing:**Stemming** is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word.E.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem". This helps in achieving the training process with a better accuracy. **Lemmatizing** is the process of grouping together the inflected forms of a word so they can be analysed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

- # Hardware and Software Tools

The libraries and packages we have used on this projected are listed below:

- Data Processing- Numpy(numerical data wrangling), Pandas(data analysis)
- Data Visualization- Matplotlib, Seaborn (graphical representations)
- Text Processing- RegEx and Natural Language Toolkit libraries

Model/s Development and Evaluation

- Possible problem-solving approaches (methods)

The target labels contains various categorical features such as 'malignant', 'highly_malignant', 'loathe', 'rude', 'abuse', 'threat'. Hence, the logical approach to building a suitable prediction model is to use regression models such as logistic regression, RFs, GBs, DTRs, KNN, XGB, etc.

- Testing of Identified Approaches (Algorithms)
  - Logistic Regression
  - Decision Tree Classifier
  - Random Forest Classifier
  - Ada Boost Classifier
  - K Neighbors Classifier
  - Extreme Gradient Boosting Classifier

- Performance of models

  - Logistic Regression:
    This is a machine learning algorithm for classification purposes. Using this, the probabilities describing the likely outcomes of a single trial are modelled using a logistic function. A baseline model executed gave out an accuracy score of 95.53%.

```
# 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 0.9595609629450577
Test accuracy is 0.9553392379679144
[[42729   221]
 [ 1917  3005]]
              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
```

- Decision Tree Classifier:
  Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.
  A simple run gave out an accuracy score of **94.00%**.

```
# 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 0.9400902406417112
[[41622  1328]
 [ 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
```

- Random Forest Classifier:
  This classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and control over-fitting. A single run of this gave an accuracy score of **95.55**%.

```
#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 0.9988540631518635
Test accuracy is 0.9554645721925134
[[42418   532]
 [ 1600  3322]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.86      0.67      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
```

- Ada Boost Classifier:

  Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996. It combines multiple classifiers to increase the accuracy of classifiers.
  A sample run of this gave an accuracy score of **95.55**%.

```
#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
```

- K Neighbors Classifier:

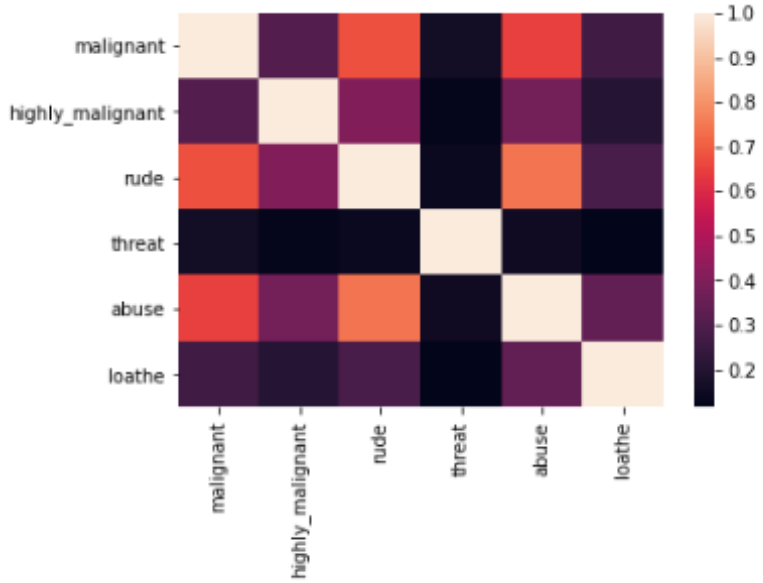
  K-nearest neighbors (k-NN) is a pattern recognition algorithm that uses training datasets to find the *k* closest relatives in future examples.
  Initial run of this classifier resulted in accuracy of **91.74**%

```
#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))
```

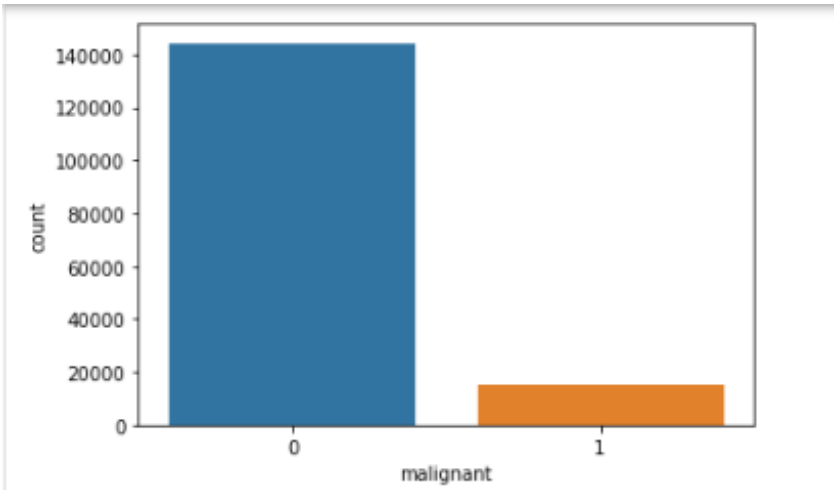
```
Training accuracy is 0.922300110117369
Test accuracy is 0.9173629679144385
[[42809   141]
 [ 3815  1107]]
              precision    recall  f1-score   support

           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
```

- Visualizations

Correlation between the multiple labels

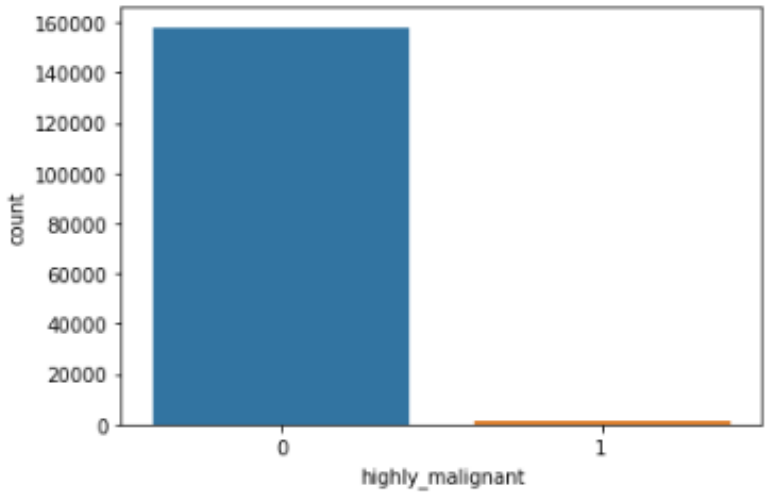


Count Plot for Malignant Label

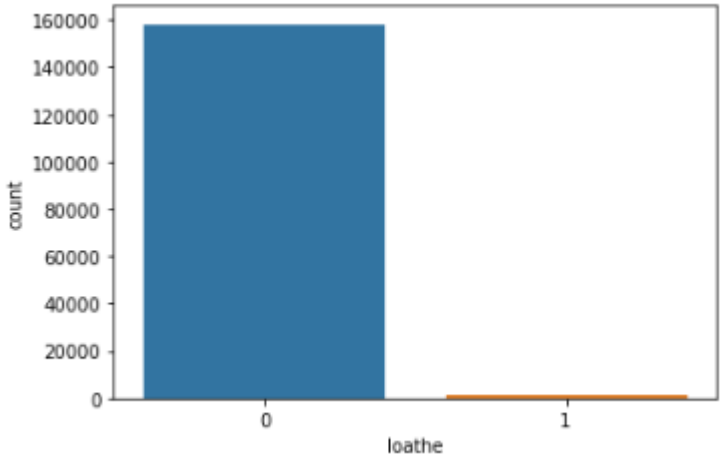


highly_malignant

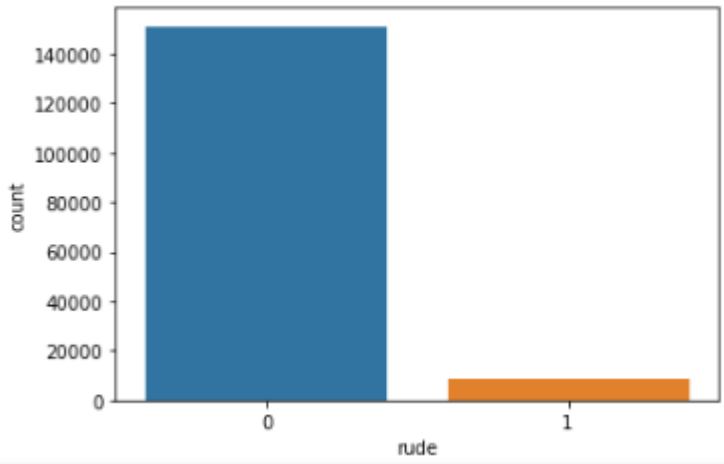
Count Plot for Highly Malignant Label



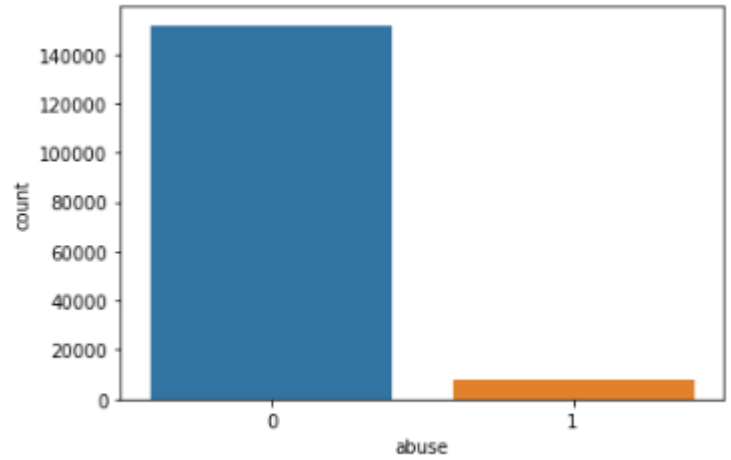
Count Plot for Loathe Label



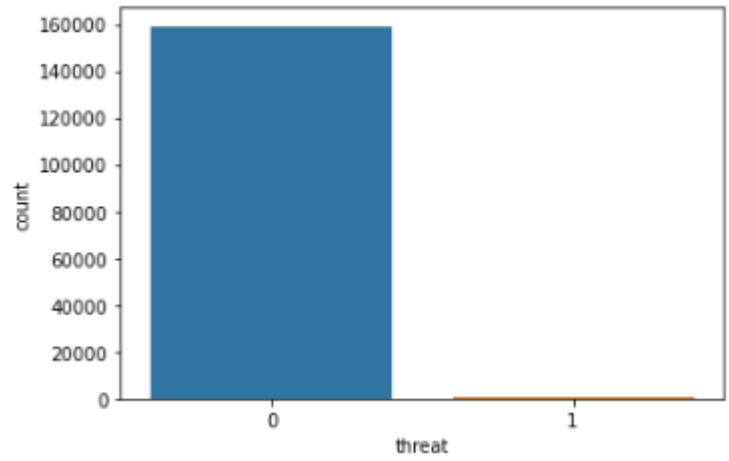
Count Plot for Rude Label



Count Plot for Abuse Label

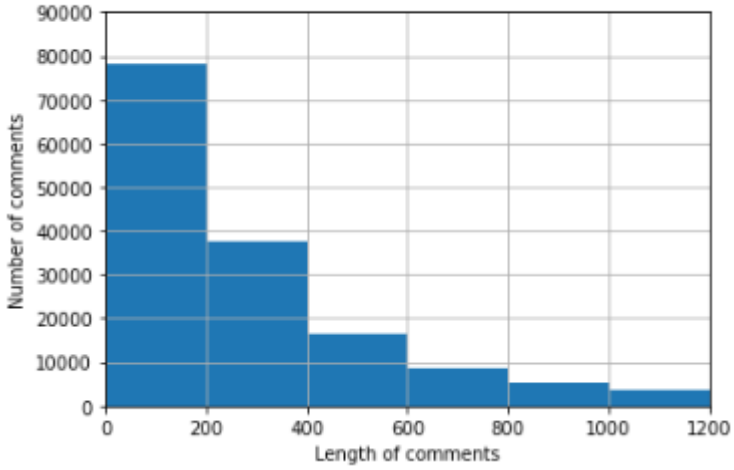


Count Plot for Threat Label



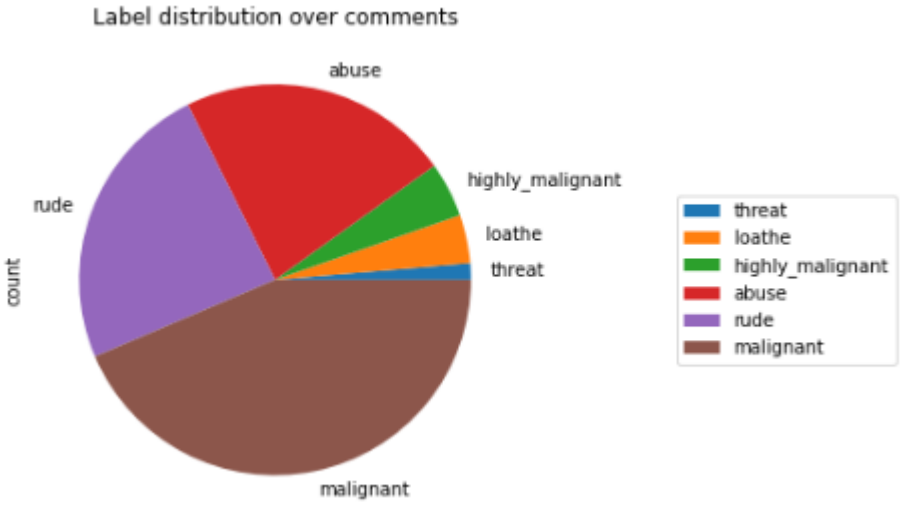
Length of Comments

```
average length of comment: 394.139
```



Word Cloud

Label Distribution over Comments



CONCLUSION

- Key Findings and Conclusions of the Study

This is the summary of the things followed in this project:

→ The first step involved collecting data and deciding what part of it is suitable for training : This step was extremely crucial since including only very small length comments would give poor results if the length was increased whereas including very long length comments would increase the number of words drastically.

→ The second major step was performing cleaning of data including punctuation removal, stop word removal, stemming and lemmatizing: This step was also crucial since the occurrence of similar origin words but having different spellings will intend to give similar classification, but computer cannot recognize this on its own. Hence, this step helped to a large extent in both removing and modifying existing words.

→ The third step was choosing models to train on: With a wide variety of classifier models, selecting which all models to train and test took lots of efforts.

.

- Scope for Future Work

The current project predicts the type or toxicity in the comment. We are planning to add the following features in the future:

→ Analyse which age group is being toxic towards a particular group or brand.

→ Add feature to automatically sensitize words which are classified as toxic.

→ Automatically send alerts to the concerned authority if threats are classified as severe.

→ Build a feedback loop to further increase the efficiency of the model.

→ Handle mistakes and short forms of words to get better accuracy of the result.