MALIGNANT COMMENTS CLASSIFIER

Submitted by: **SHANTY EMERSON**

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to FLIP ROBO TECHNOLOGIES. as well as our SME Shubham Yadav who gave me the golden opportunity to do this wonderful project on the topic Project Malignant Comment Classification which also helped me in doing this project.

INTRODUCTION

Business Problem Framing

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 behavior.

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.

Solution is we will 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.

From this model we can be predict whether the comment is 'malignant','highly_malignant','rude','threat','abuse','loathe'.

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.

Analytical Problem Framing

Methodology represents a description about the framework that is undertaken. It consists of various milestones that need to be achieved in order to fulfil the objective. We have various attributes that contribute classifying malignant and abusive comments.

The following steps represents stepwise tasks that need to be completed:

- Data Loading
- Data Cleaning
- Data pre-processing
- Data Visualization
- Model Building
- Model Evaluation
- Saving the best model

Data Sources and their formats

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes 'Id', 'Comments', 'Malignant', 'Highly malignant', 'Rude', 'Threat', 'Abuse' and 'Loathe'.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes:

1.Malignant: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
2.Highly Malignant: It denotes comments that are highly malignant and hurtful.
3.Rude: It denotes comments that are very rude and offensive.
4.Threat: It contains indication of the comments that are giving any threat to someone.
5.Abuse: It is for comments that are abusive in nature.

6.Loathe: It describes the comments which are hateful and loathing in nature.
7.ID: It includes unique Ids associated with each comment text given.
8.Comment text: This column contains the comments extracted from various social media platforms.

The dataset shape is

Train.csv◊ (159571, 8)

Test.csv◊ (153164, 2)

```
#Read the csv file into dataframe df
df = pd.read_csv("malignant_train.csv")#Importing the training data

test=pd.read_csv("malignant_test.csv")#Importing the test data
print(df.shape)
print(test.shape)
```

```
(159571, 8)
(153164, 2)
```

```
#List the fields in our dataframe
df.dtypes
```

```
id                 object
comment_text       object
malignant           int64
highly_malignant    int64
rude                int64
threat              int64
abuse               int64
loathe              int64
dtype: object
```

Data Pre-processing

Data pre-processing is a process of transforming the raw, complex data into systematic understandable knowledge.

- Data Sample: Train Data

```
df.head()#training data
```

	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 Sample: Test Data

```
test.head()#test data
```

	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.

Target Variable: Target Variable of data samples contain 8 fields which includes 'Id', 'Comments', 'Malignant', 'Highly malignant', 'Rude', 'Threat', 'Abuse' and 'Loathe'.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels.

Data Preprocessing

In order to prepare the text data for the model building we perform text preprocessing. It is the very first step of NLP projects. Some of the preprocessing steps are:

1. Making of corpus by GENERIC NLP FILTER CODE
   a) Convert all cases to lower
   b) Remove punctuations
   c) Remove Stopwords
   d) Stemming and Lemmatising

2.Applying embedding technique. (TF-IDF Encoding.)

3.Train_test_split training data(embedded_docs)in a different way as our target feature('labels') contains 6 classes.
#Our data is now ready for model building process input i.e vectors(numerical format)

Data Preprocessing work:

Checking for missing values:

```
#Checking missing values in train data
df.isnull().sum()
```

```
id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
```

```
#Checking missing values in test data
test.isnull().sum()
```

```
id              0
comment_text    0
dtype: int64
```

Fortunately, no missing values are found in both Train and Test Data.

Data Cleaning:

We define a function that replaces all the non-alpha characters with space, replacing brackets, removing html tags, replacing next line spaces, replacing multiple line spaces, extra spaces and for characters like aren't replacing n't with not. Finally converting to lower case.

```
def clean_text(data):
    for i in range(len(df)):

        #replacing all non alpha charcaters with space
        data = re.sub('[^a-zA-Z]', ' ',data)

        #replacing brackets
        data = re.sub('\[|\]', ' ', data)

        #removing html tags
        data = re.sub('https?://\S+|www\.\S+', '', data)

        #replacing next line spaces
        data = re.sub('\n', '', data)

        #replacing multiple line spaces and _ with single space bar only
        data = re.sub(' +|_+', ' ', data)

        #for characters like aren't I will replace n't with not
        data = re.sub('n\st',' not',data)

        #Lower case
        data = data.lower()
        return data
```

```
# Cleaning Train data using above function.
df['comment_text']=df['comment_text'].apply(clean_text)
```

```
# Cleaning Test data using above function.
test['comment_text']=test['comment_text'].apply(clean_text)
```

Preprocessing train data, tokenizing and Lemmatizing.

```
corpus_train = []

def pre_process_train(start,end, data):
    for i in range(start,end):
        review = data['comment_text'][i]
        #from each row of message only keeping element which starts with albhabet
        review = word_tokenize(review)
        words = []
```

```
    for word in review:
        if len(word)>2:
            words.append(word)
    review_length = words

    word_lemmatize = []
    for wrd in review_length:
        if wrd not in set(stopwords.words('english')):
            word_lemmatize.append(WordNetLemmatizer().lemmatize(wrd))

    review_lemmatise = word_lemmatize
    review = ' '.join(review_lemmatise)

    corpus_train.append(review)
```

```
# Applying function to train data.
pre_process_train(0, 159571, df)
```

Same way Preprocessing is done to Test data as well and cleaned comment column is saved and reassigned to the Comments column for both Train and Test data.

```
# Saving filtered/cleaned comments to a new variable
clean_comment = pd.DataFrame(corpus_train)
```

```
# Reassigning data to clean_comment column.
df['clean_comment'] = clean_comment
df.head()
```

Deleted Unnecessary columns from both data.

```
# Deleting id column.
df.drop('id', axis = 1, inplace = True)
```

Train Data after Preprocessing:

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	clean_comment
0	explanation why the edits made under my userna...	0	0	0	0	0	0	explanation edits made username hardcore metal...
1	d aww he matches this background colour i m se...	0	0	0	0	0	0	aww match background colour seemingly stuck th...
2	hey man i m really not trying to edit war it s...	0	0	0	0	0	0	hey man really trying edit war guy constantly ...
3	more i ca not make any real suggestions on im...	0	0	0	0	0	0	make real suggestion improvement wondered sect...
4	you sir are my hero any chance you remember wh...	0	0	0	0	0	0	sir hero chance remember page

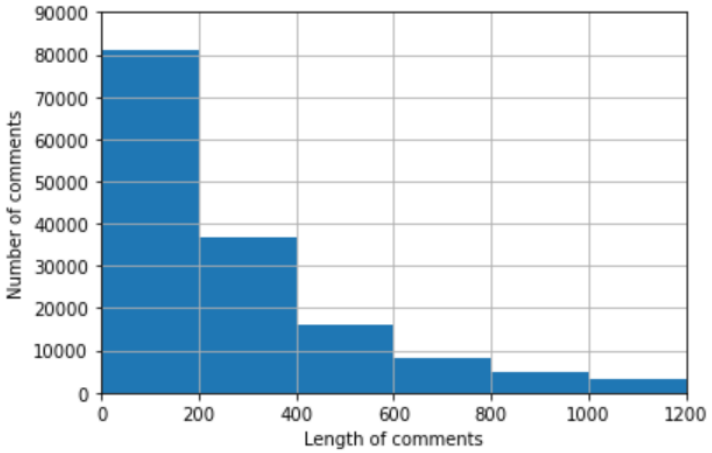
Test Data after Preprocessing:

	comment_text	clean_comment
0	yo bitch ja rule is more succesful then you ll...	bitch rule succesful ever whats hating sad mof...
1	from rfc the title is fine as it is imo	rfc title fine imo
2	sources zawe ashton on lapland	source zawe ashton lapland
3	if you have a look back at the source the inf...	look back source information updated correct f...
4	i do not anonymously edit articles at all	anonymously edit article

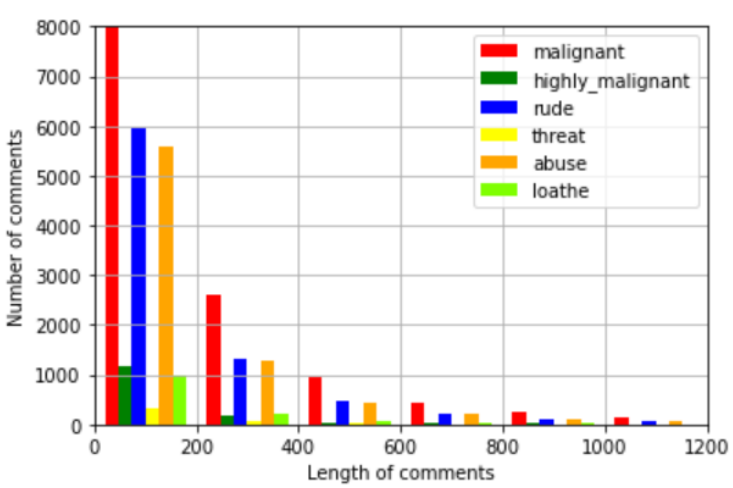
Data Visualization and Analysis

Analyzing No. of comments for train data having lengths varying from 0 to 1200.

```
average length of comment: 374.681
```



Number of comments classified as malignant, highly_malignant, rude, ...etc. depending on their lengths for train data.

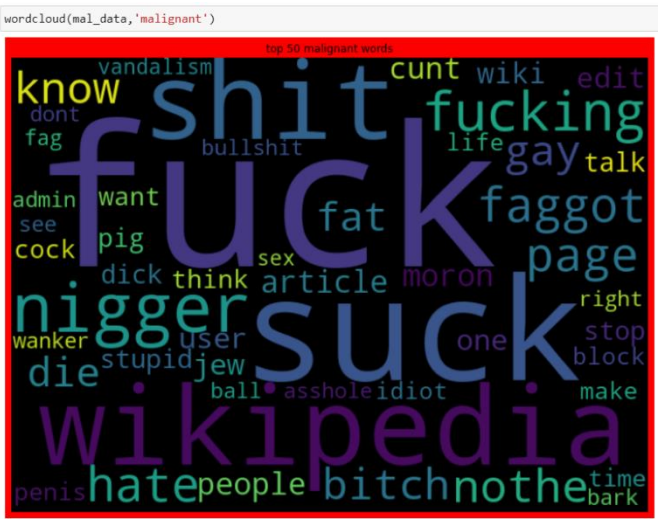


Inference from Data: Data with low sentence lengths (<200 are Malignant, highly Malignant and Rude.

Some very large length comments can be seen, in our dataset. These pose serious problems like adding excessively more words to the training dataset, causing training time to increase and accuracy to decrease! Hence, a threshold of 400 characters will be created and only comments which have length smaller than 400 will be used further.

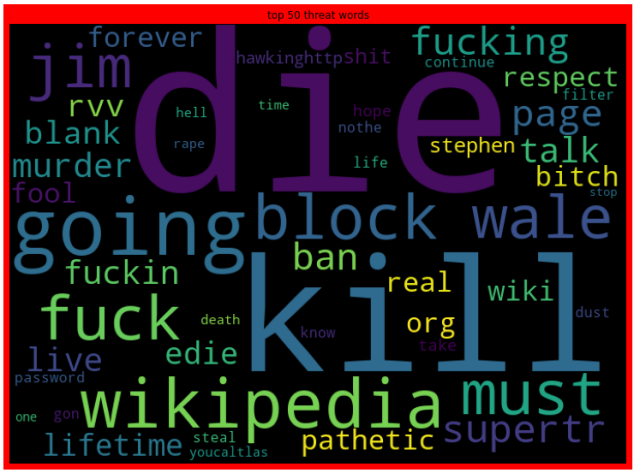
Data Visualization using WordCloud:

Word clouds or tag clouds are graphical representations of word frequency that give greater prominence to words that appear more frequently in a source text.

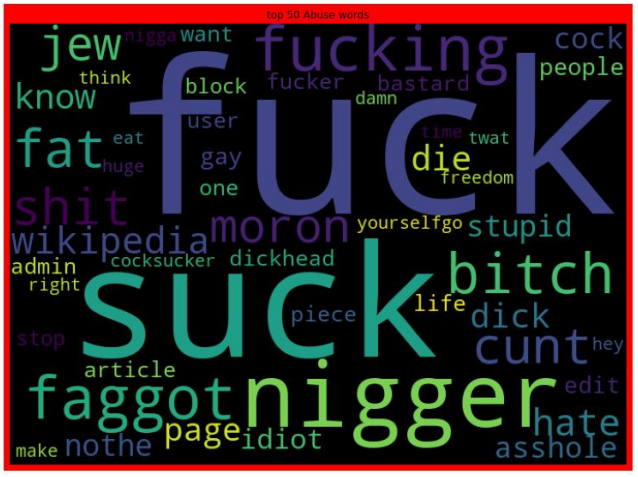


```
wordcloud(rude_data,'Rude')
```


```
wordcloud(threat_data,'threat')
```




```
wordcloud(abuse_data,'Abuse')
```




```
wordcloud(loathe_data,'Loathe')
```


```
#Saving both preprocessed files.
df.to_csv('final_train.csv')
test.to_csv('final_test.csv')
```

DATA PREPARATION

```
# Uploading preprocessed data and dropping unnecessary column.
data = pd.read_csv('final_train.csv')
data.drop('Unnamed: 0',axis=1,inplace=True)
data.head()
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	clean_comment
0	explanation why the edits made under my userna...	0	0	0	0	0	0	explanation edits made username hardcore metal...
1	d aww he matches this background colour i m se...	0	0	0	0	0	0	aww match background colour seemingly stuck th...
2	hey man i m really not trying to edit war it s...	0	0	0	0	0	0	hey man really trying edit war guy constantly ...
3	more i ca not make any real suggestions on im...	0	0	0	0	0	0	make real suggestion improvement wondered sect...
4	you sir are my hero any chance you remember wh...	0	0	0	0	0	0	sir hero chance remember page

Checking for Null Values again after Preprocessing, and dropped all NA values.

```
# Checking null values.
data.isna().sum()
```

```
comment_text          0
malignant             0
highly_malignant      0
rude                  0
threat                0
abuse                 0
loathe                0
clean_comment       129
dtype: int64
```

```
# Removing all null values.
data.dropna(inplace = True)
```

Vectorizing with TF-IDF VECTORIZER

Text data requires special preparation before you can start using it for predictive modeling. The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).

The Tfidf Vectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

```
# Vectorizing with TDIDF vectorizer.
tfidf = TfidfVectorizer()

feature1 = tfidf.fit_transform(data['clean_comment'])
```

Train Test Split

Train_test_split training data (embedded docs) in a different way as our target feature('labels') contains 6 classes. Our data is now ready for model building process input ie. vectors (numerical format).

```
# Assigning data for modelling.
X = feature1
y = data['malignant']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 42)
```

Model Building and Evaluation:

Building the model with all the features.

Algorithms Used:

1.Multinomial Naive Bayes Classifier (The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification)

2. Complement Naive Bayes Classifier (This approach is almost the same as the Multinomial, though now we count the occurrences of a word in the complement to the class. For example, for the spam message we will count the repetitions of each word in all the non-spam messages.)

3. AdaBoost Classifier

Key Metrics for success in solving problem under consideration.

This is a text multiclass classification problem and I have chosen.

1. Accuracy
2. Log-loss
3. Hamming loss as my evaluation metrics

MODEL1: Multinomial Naive Bayes Classifier

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
y_pred = model.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     43238
           1       0.99      0.18      0.30      4595

    accuracy                           0.92     47833
   macro avg       0.95      0.59      0.63     47833
weighted avg       0.93      0.92      0.89     47833
```

```
evaluate_score(y_test, y_pred)
```

```
Hamming_loss : 7.9380344113896255
Accuracy : 92.06196558861038
Log_loss : 2.7417001226653186
```

MODEL2: Complement Naive Bayes Classifier

```
model2 = ComplementNB()
model2.fit(X_train, y_train)
```

```
ComplementNB()
```

```
y_preds = model2.predict(X_test)
```

```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96     43238
           1       0.71      0.47      0.57      4595

    accuracy                           0.93     47833
   macro avg       0.83      0.72      0.76     47833
weighted avg       0.92      0.93      0.92     47833
```

```
evaluate_score(y_test, y_preds)
```

```
Hamming_loss : 6.909455815023101
Accuracy : 93.09054418497689
Log_loss : 2.3864559872086595
```

MODEL3: AdaBoost Classifier

```
model3 = AdaBoostClassifier(ComplementNB())
model3.fit(X_train, y_train)
```

```
AdaBoostClassifier(base_estimator=ComplementNB())
```

```
y_pred_ada = model3.predict(X_test)
```

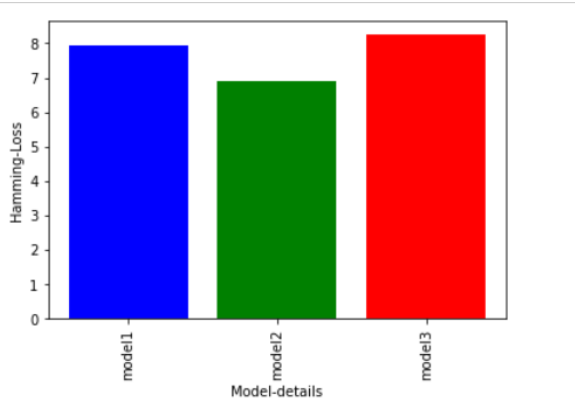
```
print(classification_report(y_test, y_pred_ada))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     43238
           1       0.88      0.16      0.28      4595

    accuracy                           0.92     47833
   macro avg       0.90      0.58      0.62     47833
weighted avg       0.91      0.92      0.89     47833
```

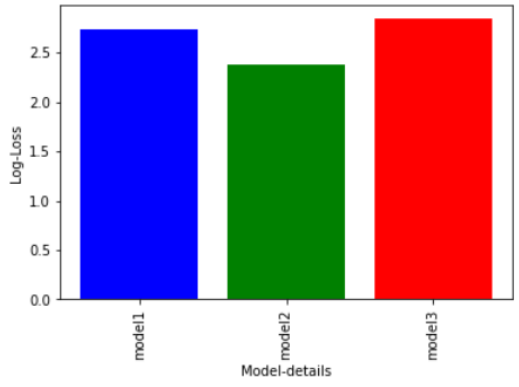
```
evaluate_score(y_test, y_pred_ada)
```

```
Hamming_loss : 8.247444233060857
Accuracy : 91.75255576693915
Log_loss : 2.8485679768793144
```

Hamming loss comparison of all models:



Log loss comparison of all models:



If we compare all the models on basis of hamming-loss :

The hamming loss (HL) is defined as the fraction of the wrong labels to the total number of labels. For a multi-label problem, we need to decide a way to combine the predictions of the different labels to produce a single result. The method chosen in hamming loss is to give each label equal weight.

The best model would be Complement Naive Bias Classifier model. It has a hamming-loss of 6.90 % only.

If we compare all the models on basis of Log-loss :

Log Loss quantifies the accuracy of a classifier by penalizing false classifications. The exponentially decaying curve it possesses clearly indicates the same.

The best model will be the Complement Naive Bias Classifier model. It has a Log-loss of 2.38% only.

Saving The Best Model

```
import joblib
joblib.dump(model2, 'malginant-classification.pkl')
```

```
['malginant-classification.pkl']
```

Hardware and Software Requirements and Tools Used

Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

1. Lenovo flex corei5 laptop

2. Jupyter Notebook :The Jupyter Notebook is an interactive environment for running code in the browser. It is a great tool for exploratory data analysis and is widely used by data scientists.

3. MS PowerPoint: For preparing presentation of project.

4. MS word: For preparing report

5. Pandas: is a software library written for the Python programming language for data manipulation and analysis.

6. Numpy: is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

7. Matplotlib: is a plotting library for the Python programming language and its numerical mathematics extension

8. NumPy.Seaborn: is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

9. Sklearn: Scikit-learn (formerly scikits. learn and also known as sklearn) is a free software machine learning library for the Python programming language.

   Libraries used:-
   1. numpy
   2. pandas
   3. matplotlib
   4. sweetviz
   5. seaborn
   6. sklearn
   7. itertools
   8. Seaborn

CONCLUSION

The final step was to figure out which model gave the best results. A mixture of hamming-loss and log-loss was used to select the same.

- Key Findings and Conclusions of the Study
  - ❖ Data with low sentence lengths (<200) are malignant, rude, abuse.
  - ❖ Some very large length comments can be seen, in our dataset. These pose serious problems like adding excessively more words to the training dataset, causing training time to increase and accuracy to decrease.