NAME OF THE PROJECT

Malignant Comments Classifier - Multi Label Classification Project Using NLP

Submitted by:

Ms. Yashshree Baviskar

FLIPROBO SME:

Mr. Mohd Kashif

ACKNOWLEDGMENT

I would like to express my special gratitude to "Flip Robo" team, who has given me this opportunity to deal with a beautiful dataset and it has helped me to improve my analyzation skills. And I want to express my huge gratitude to Mr. Mohd kashif (SME Flip Robo), she is the person who has helped me to get out of all the difficulties I faced while doing the project.

A huge thanks to "Data trained" who are the reason behind my Internship at Fliprobo. Last but not least my parents who have been my backbone in every step of my life.

References use in this project:

1. SCIKIT Learn Library Documentation
2. Blogs from towardsdatascience, Analytics Vidya, Medium
3. Andrew Ng Notes on Machine Learning (GitHub)
4. Data Science Projects with Python Second Edition by Packt
5. Hands on Machine learning with scikit learn and tensor flow by Aurelien Geron
6. Thomas A Birkland. An introduction to the policy process: Theories, concepts, and models of public policy making. Routledge, 2019.
7. Mohammad, Fahim. "Is pre processing of text really worth your time for online comment classification?." arXiv preprint arXiv:1806.02908 (2018).
8. Yin, Dawei, et al. Detection of harassment on web 2.0 Proceedings of the Content Analysis in the WEB 2 (2009):1-7
9. Hamida, Chady Ben, Victoria Ge, and Nolan Miranda. "Toxic Comment Classification and Unintended Bias." (2019).
10. Chakrabarty, Navoneel. "A Machine Learning Approach to Comment Toxicity Classification." Computational Intelligence in Pattern Recognition. Springer, Singapore, 2020. 183-193.
11. Nadine Farag, Samir Abou El-Seoud, Gerard McKee, and Ghada Hassan. Bullying hurts: A survey on non-supervised techniques for cyber-bullying detection. In Proceedings of the 2019 8th

International Conference on Software and Information Engineering, pages 85–90, 2019

12. Sweta Karlekar and Mohit Bansal. Safecity: Understanding diverse forms of sexual harassment personal stories. arXiv preprint arXiv:1809.04739, 2018.
13. Revati Sharma, Meetkumar Patel, "Toxic Comment Classification Using Neural Networks and Machine Learning", Vol. 5, Issue 9, September 2018, DOI 10.17148/IARJSET.2018.597, pg no:47- 52
14. Chu, T & Jue, K. Comment Abuse Classification with Deep Learning

Chap 1. Introduction

1.1 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. 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.

1.2 Conceptual Background of the Domain Problem

Online forums and social media platforms have provided individuals with the means to put forward their thoughts and freely express their opinion on various issues and incidents. In some cases, these online comments contain explicit language which may hurt the readers. Comments containing explicit language can be classified into

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

To protect users from being exposed to offensive language on online forums or social media sites, companies have started flagging comments and blocking users who are found guilty of using unpleasant language. Several Machine Learning models have been developed and deployed to filter out the unruly language and protect internet users from becoming victims of online harassment and cyberbullying

Now-a-days users leave numerous comments on different social networks, news portals, and forums. Some of the comments are toxic or abusive. *Due to numbers of comments, it is unfeasible to manually moderate them, so most of the systems use some kind of automatic discovery of toxicity using machine learning models.*

Based on the report by Birkland [6], **'Twitter users generate 500 million tweets per day, and in 2019 they had a 14% year-over-year growthof daily usage'.** However, behind the shield of computers as virtual walls, some individuals also think they can abuse and harass other people's opinions and characters. Accordingly, a jargon word has been coined recently to address such behaviours as "cyberbullying". Based on U.S. Department of Health and Human Services, cyberbullying could be introduced as mistreatments that occurs all over digital instruments and devices such as computers, tablets, and mobile phones. Cyberbullying may take place via short massage system (SMS), general apps with the possibility of communication between users, online social media forums, or even online gaming where individuals can virtually participate in. *Cyberbullying includes posting, sending, or sharing harmful, negative, mean, or false content about someone else either directly sent to theperson or post as a general comment where other could observe it.* It also includes sharing private or personal info about someone else in order to humiliation or embarrassment.

Detecting Toxic comments has been a great challenge for the all the scholars in the field of research and development. This domain has drawn lot of interests not just because of the spread of hate but also people refraining people from participating in online forums which diversely affects for all the creators/content-providers to provide a relief

to engage in a healthy public interaction which can be accessed by public without any hesitation. **There have been sure turns of developments in this area which includes couple of models served through API. But the models still make errors and still fail to provide an accurate solution to the problem.**

1.3 Review of Literature

Sentiment classification regarding toxicity has been intensively researched in the past few years, largely in the context of social media data where researchers have applied various machine learning systems to try and tackle the problem of toxicity as well as the, more well-know, task of sentiment analysis. Comment abuse classification research initially began with Yin et. al. [8] application of combining TF-IDF with sentiment/contextual features. They compared the performance of this model with a simple TF-IDF model and reported a 6% increase in F1 score of the classifier on chat style datasets.

Fahim Mohammad et. al. [7] proposed a logistic regression, Bi-LSTM, XGBoost and naive bayes SVM Approaches for text analytics in toxicity classification using n-gram feature extraction technique, obtaining best Accuracy of 80% using NB SVM and Bi-LSTM. In that, did not tune the parameters of different algorithms presented in their experiment.

Chady Ben Hamida et. al. [9] applied various Deep Learning and Machine Learning approaches CNN, logistic regression and naive bayes for the task of detect toxic comments and find the bias, obtaining a Label Accuracy of 94.84% using CNN classification technique after extract feature via Glove embedding method. Future work of particular paper is to improve pre-processing steps and would apply recurrent neural network with BLSTM.

Navoneel Chakrabarty et al. [10] proposed a Machine Learning Approach involving Decision Tree Classifier with TF-IDF and bag-of-word feature generation technique for comment toxicity classification, obtaining a Mean Accuracy of 91.64%.

Farag, El-Seoud et. al. [11] reported that extensive numbers of literature have shown that supervised learning techniques have been the most frequently used methods for cyber-bullying detection.

Nevertheless, other non-supervised techniques and methods have recognized to be operative on cyber-bullying recognition. Also, Karlekar and Bansal et. al. [12] reported an increased number of personal sexual harassment and abuse that are shared and posted online. In this study, authors presented the task of automatically categorizing and analysing various forms of sexual harassment, based on stories shared on the online forum SafeCity and used labelling levels of groping, ogling, and commenting; their results indicated that single-label CNN-RNN model achieves an accuracy of 86.5.

In 2018 Revati Sharma et. al. [13] performed classifying toxic comments using Neural networks like CNN and RNN, Using the word embedding techniques and also performing a head on comparison with the primary level neural network algorithms, results with intricate Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN). Long-Short Term Memory (LSTM) results, the obtained analysis show that the **LSTM performing a way that is better than the CNNs in terms of both the precision and time execution given the same number of epoch** and hence are preferable to use rather than CNN with word-level embeddings.

Chu and Jue et. al. [14] compared the performance of various deep learning approaches to this problem, specifically using both word and character embeddings. They assessed the performance of recurrent neural networks with LSTM and word embeddings, a CNN with word embeddings and a CNN with character embeddings. The best performance they achieved was a 93 % accuracy using the character level CNN model.

1.4 Motivation for the Problem Undertaken

The project is provided to me by Flip Robo Technologies as a part of the internship programme. The exposure to real world data and the opportunity to deploy my skillset in solving a real time problem has been the primary motivation.

Detecting Toxic comments has been a great challenge for the all the scholars in the field of research and development. This domain has drawn lot of interests not just because of the spread of hate but also people refraining people from participating in online forums which

diversely affects for all the creators/content-providers to provide a relief to engage in a healthy public interaction which can be accessed by public without any hesitation. There have been sure turns of developments in this area which includes couple of models served through API. But the models still make errors and still fail to provide an accurate solution to the problem.

Chap 2 Analytical Problem Framing

1. Mathematical / Analytical Modelling of the Problem

In order to apply text classification, the unstructured format of text has to be converted into a structured format for the simple reason that it is much easier for computer to deal with numbers than text. This is mainly achieved by projecting the textual contents into Vector Space Model, where text data is converted into vectors of numbers. For this purpose, various operations are performed on dataset which we will see in further section on data pre-processing in this report.

2. 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.

```
# Importing dataset excel file using pandas.
df=pd.read_csv('train.csv')
```

```
print('No. of Rows :',df.shape[0])
print('No. of Columns :',df.shape[1])
pd.set_option('display.max_columns',None) # This will enable us to see truncated columns
df.head()
```

```
No. of Rows : 159571
No. of Columns : 8
```

	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

The different features in dataset are describe as below table :-

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.

3. Data Pre-processing

The dataset is large and it may contain some data error. In order to reach clean, error free data some data cleaning & data pre-processing performed data.

- We first convert the comments to lower-case and then use custom made functions to remove *html-tags*, *punctuation and non-alphabetic characters* from the comments.
- Next, we remove all the *stop-words* present in the comments using the default set of stop-words that can be downloaded from *NLTK* library. We also add few stop-words to the standard list.

```
#Defining the stop words
stop_words = stopwords.words('english')

#Defining the lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
#Replacing '\n' in comment_text
df['comment_text'] = df['comment_text'].replace('\n',' ')
```

- Stop words are basically a set of commonly used words in any language, not just English. The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead.
- Next, we do stemming. There exist different kinds of stemming which basically transform words with roughly the same semantics to one standard form. For example, for amusing, amusement, and amused,the stem would be amus.

To perform all above operation one function is created.

```
#Function Definition for using regex operations and other text preprocessing for getting cleaned texts
def clean_comments(text):

    #convert to lower case
    lowered_text = text.lower()

    #Replacing email addresses with 'emailaddress'
    text = re.sub(r'^.+@[^\.].*\.[a-z]{2,}$', 'emailaddress', lowered_text)

    #Replace URLs with 'webaddress'
    text = re.sub(r'http\S+', 'webaddress', text)

    #Removing numbers
    text = re.sub(r'[0-9]', " ", text)

    #Removing the HTML tags
    text = re.sub(r"<.*?>", " ", text)

    #Removing Punctuations
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\_',' ',text)

    #Removing all the non-ascii characters
    clean_words = re.sub(r'[^\x00-\x7f]',r'', text)

    #Removing the unwanted white spaces
    text = " ".join(text.split())

    #Splitting data into words
    tokenized_text = word_tokenize(text)

    #Removing remaining tokens that are not alphabetic, Removing stop words and Lemmatizing the text
    removed_stop_text = [lemmatizer.lemmatize(word) for word in tokenized_text if word not in stop_words if word.isalpha()]

    return " ".join(removed_stop_text)
```

Finally, created function clean_comments() is executed on train & test dataset.

```
# Calling the above function for the column comment_text in training dataset to replace original with cleaned text
df['comment_text'] = df['comment_text'].apply(clean_comments)
df['comment_text'].head()
```

```
0    explanation edits made username hardcore metal...
1    aww match background colour seemingly stuck th...
2    hey man really trying edit war guy constantly ...
3    make real suggestion improvement wondered sect...
4                      sir hero chance remember page
Name: comment_text, dtype: object
```

- After splitting the dataset into train & test sets, we want to summarize our comments and convert them into numerical vectors.
- One technique is to pick the most frequently occurring terms (words with high term frequency or TF). However, the most frequent word is a less useful metric since some words like '*this*', '*a*' occur very frequently across all documents.
- Hence, we also want a measure of how unique a word is i.e., how infrequently the word occurs across all documents (inverse document frequency or IDF).

- So, the product of TF & IDF (TF-IDF) of a word gives a product of how frequent this word is in the document multiplied by how unique the word is w.r.t. the entire corpus of documents.

- 1. Vectorizer & Spliting Train dataset

```
# Converting the features into number vectors
tf_vec = TfidfVectorizer(max_features = 2000, stop_words='english')
```

```
# Let's Separate the input and output variables represented by X and y respectively in train data and convert them
X = tf_vec.fit_transform(df['comment_text']).toarray()
```

```
output_labels= df.columns[1:7]
```

```
# output variables
from scipy.sparse import csr_matrix
Y = csr_matrix(df[output_labels]).toarray()

# checking shapes of input and output variables to take care of data imbalance issue
print("Input Variable Shape:", X.shape)
print("Output Variable Shape:", Y.shape)
```

```
Input Variable Shape: (159571, 2000)
Output Variable Shape: (159571, 6)
```

- Words in the document with a high TFIDF score occur frequently in the document and provide the most information about that specific document.

- 2. Vectorizer & Spliting Train dataset

```
# Doing the above process for test data
test_vec = tf_vec.fit_transform(dft['comment_text'])
test_vec
```

```
<153164x2000 sparse matrix of type '<class 'numpy.float64'>'
        with 2138199 stored elements in Compressed Sparse Row format>
```

```
test_vec.shape
```

```
(153164, 2000)
```

- *TF-IDF* is easy to compute but its disadvantage is that it does not capture position in text, semantics, co-occurrences in different documents, etc.

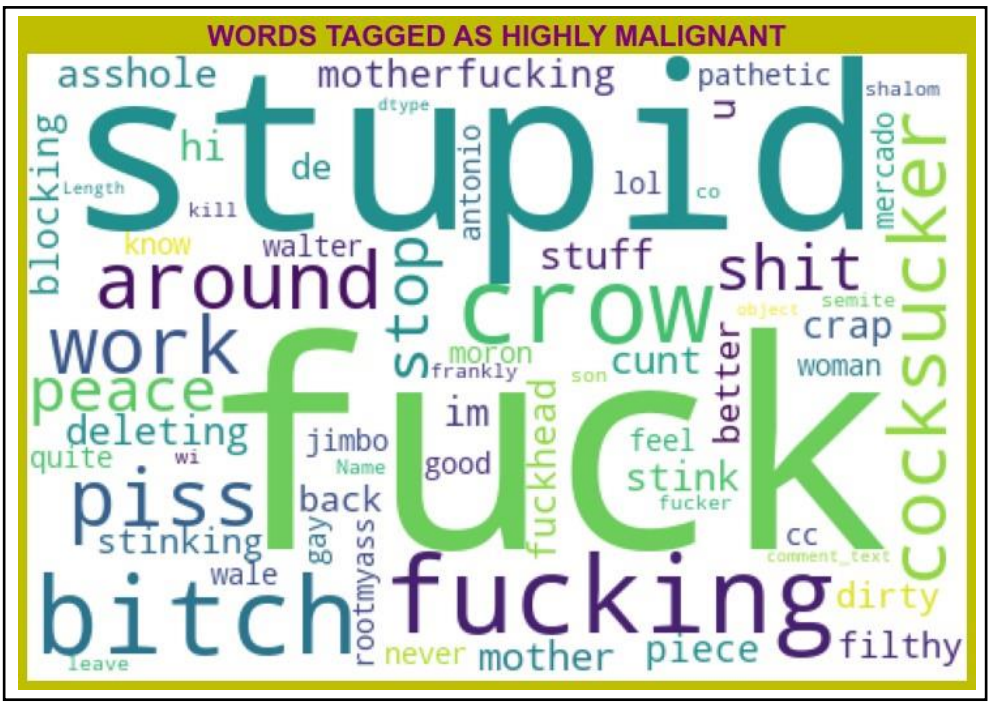
4. Data Inputs- Logic- Output Relationships

The dataset consists of features with a label. The features are independent and label is dependent as our label varies the values (text) of our independent variable's changes. Using word cloud, we can see most occurring word for different categories.

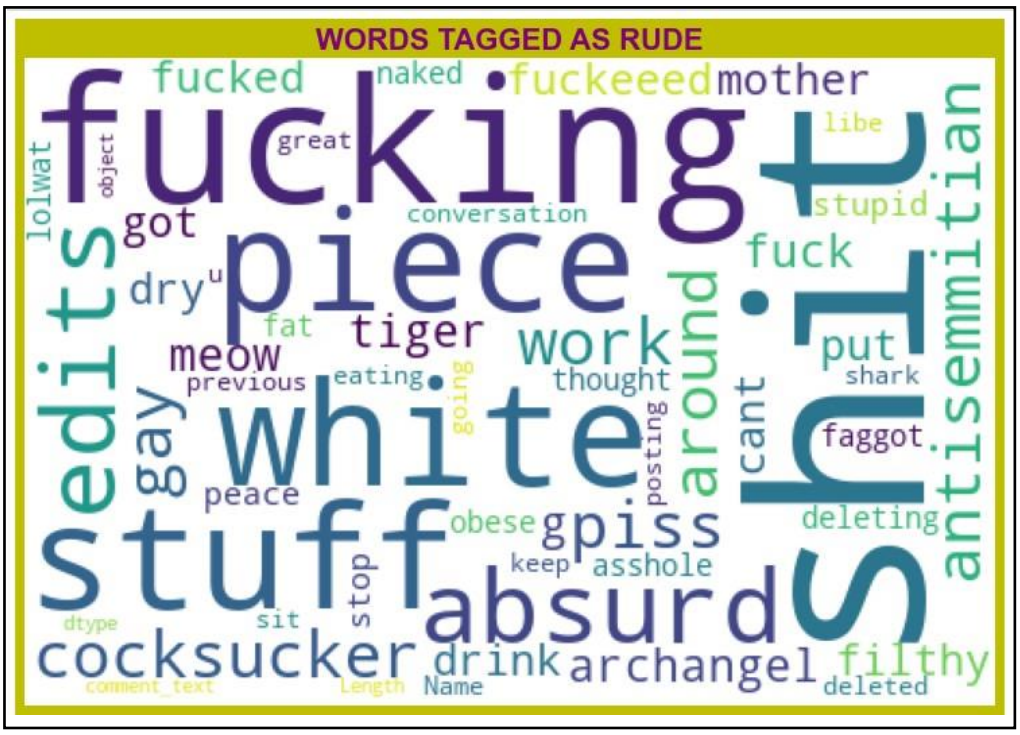
```
# Plotting for malignant
df_malignant=df[(df['malignant']==1)]
wordcloud=WordCloud(height=300,width=450,max_words=300,background_color="white").generate(str(df_malignant['comment_text']))
plt.figure(figsize=(10,10),facecolor='y')
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.title(label='WORDS TAGGED AS MALIGNANT',fontdict={'fontsize':22, 'fontweight':'bold', 'color':'purple'})
plt.show()
```

I have analysed the 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 form 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.

From word cloud of malignant comments, it is clear that it mostly consists of words like edits, hey, white, fucking, gay, cocksucker, work, think, Taliban etc.

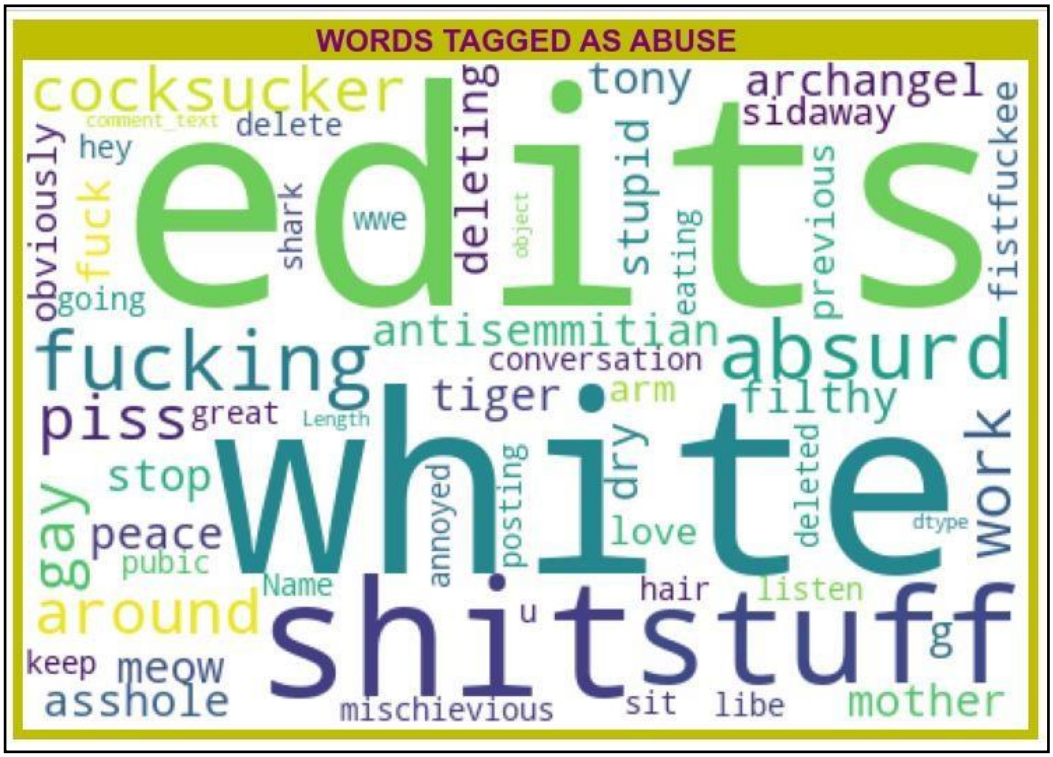


From word cloud of *Highly malignant comments*, it is clear that it mostly consists of words like fuck, stupid, fucking, bitch, crow, shit, cocksucker etc.



From word cloud of *Rude comments*, it is clear that it mostly consists of words like fucking, shit, white, piece, edits, stuff, absurd etc.

From word cloud of Threat comments, it is clear that it mostly consists of words like fuck, suck, Bitch, die, stupid, etc.



From word cloud of Abuse comments, it is clear that it mostly consists of words like edits, white, shit, stuff, fuck, piss, fucking etc.

From word cloud of *Loathe comments*, it is clear that it mostly consists of words like fuck, gay, kill, think, jew, u etc.

5. Hardware & Software Requirements with Tool Used

Hardware Used -

1. Processor — Intel i3 processor with 2.4GHZ
2. RAM — 4 GB
3. GPU — 2GB AMD Radeon Graphics

card Software utilised -

1. Anaconda – Jupyter Notebook

2. Google Colab – for Hyper parameter tuning

Libraries Used – General library for data wrangling & visualisation

```
#Importing warning library to avoid any warnings
import pandas as pd # for data wrangling purpose
import numpy as np # Basic computation library
import seaborn as sns # For Visualization
import matplotlib.pyplot as plt # ploting package
%matplotlib inline
import warnings # Filtering warnings
warnings.filterwarnings('ignore')
```

Libraries used for Text Mining / Text Analytics are:

```
#Importing required libraries
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from wordcloud import WordCloud
```

Libraries used for machine learning model building

```
#Importing Machine learning Model library
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.svm import SVC, LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import hamming_loss, log_loss
```

Chap. 3 Models Development & Evaluation

1. Identification Of Possible Problem-Solving Approaches (Methods)

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.

This problem belongs to multi label classification category which can be solve using several different approaches. Multi-label classification originated from the investigation of text categorisation problem, where each document may belong to several predefined topics simultaneously.

Difference between multi-class classification & multi-label classification is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related.

For example, multi-class classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time. Whereas, an instance of multi-label classification can be that a text might be about any of religion, politics, finance or education at the same time or none of these.

Multi-Label Classification Techniques:

1. OneVsRest

- Traditional two-class and multi-class problems can both be cast into multi-label ones by restricting each instance to have only one label. On the other hand, the generality of multi-label problems inevitably makes it more difficult to learn. An intuitive approach to solving multi- label problem is to decompose it into multiple independent binary classification problems (one per category).

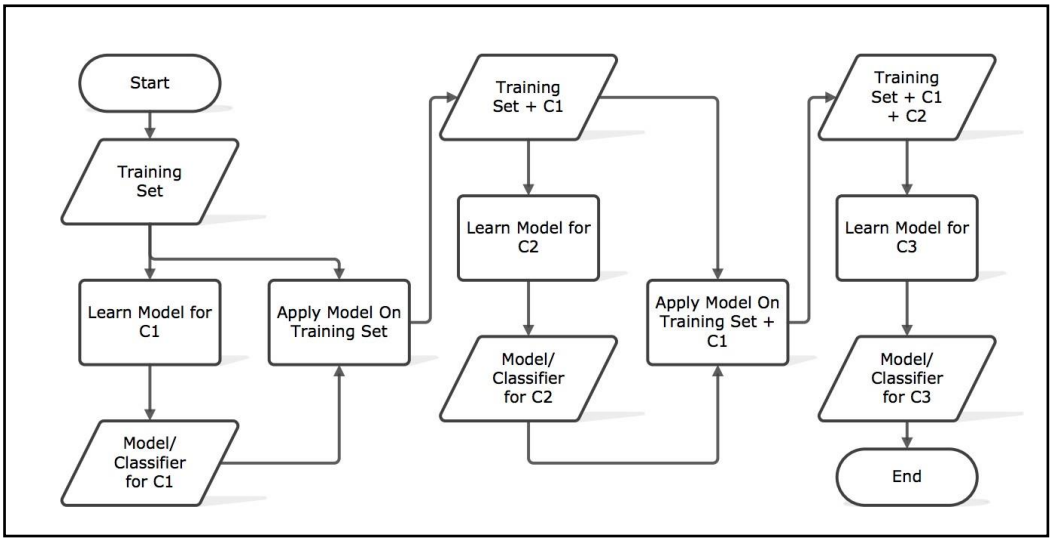
- In an "one-to-rest" strategy, one could build multiple independent classifiers and, for an unseen instance, choose the class for which the confidence is maximized.
- **The main assumption here is that the labels are mutually exclusive.** You do not consider any underlying correlation between the classes in this method.
- For instance, it is more like asking simple questions, say, "is the comment toxic or not", "is the comment threatening or not?", etc. Also, there might be an extensive case of overfitting here, since most of the comments are unlabelled, i.e., most of the comments are clean comments.

2. Binary Relevance

- In this case an ensemble of single-label binary classifiers is trained, one for each class. Each classifier predicts either the membership or the non-membership of one class. The union of all classes that were predicted is taken as the multi-label output. This approach is popular because it is easy to implement, however it also ignores the possible correlations between class labels.
- In other words, if there's q labels, the binary relevance method creates q new data sets from the images, one for each label and train single-label classifiers on each new data set. One classifier may answer yes/no to the question "does it contain trees?", thus the "binary" in "binary relevance". This is a simple approach but does not work well when there's dependencies between the labels.
- OneVsRest & Binary Relevance seem very much alike. If multiple classifiers in OneVsRest answer "yes" then you are back to the binary relevance scenario.

3. Classifier Chains

- A chain of binary classifiers C0, C1, . . ., Cn is constructed, where a classifier Ci uses the predictions of all the classifier Cj, where j < i. This way the method, also called classifier chains (CC), can take into account label correlations.
- The total number of classifiers needed for this approach is equal to the number of classes, but the training of the classifiers is more involved.
- Following is an illustrated example with a classification problem of three categories {C1, C2, C3} chained in that order.



4. Label Powerset

- This approach does take possible correlations between class labels into account. More commonly this approach is called the label-powerset method, because it considers each member of the power set of labels in the training set as a single label.
- This method needs worst case (2^|C|) classifiers, and has a high computational complexity.
- However, when the number of classes increases the number of distinct label combinations can grow exponentially. This easily leads to combinatorial explosion and thus computational infeasibility.

5. Adapted Algorithm

- Algorithm adaptation methods for multi-label classification concentrate on adapting single-label classification algorithms to the multi-label case usually by changes in cost/decision functions.
- Here we use a multi-label lazy learning approach named ML-KNN which is derived from the traditional K-nearest neighbour (KNN) algorithm.
- The skmultilearn.adapt module implements algorithm adaptation approaches to multi-label classification, including but not limited to ML-KNN.

2. Testing of Identified Approaches (Algorithms)

The different classification algorithm used in this project to build ML model are as below:

- ❖ Logistics Regression
- ❖ Random Forest classifier
- ❖ Support Vector Classifier
- ❖ AdaBoost Classifier

In this project we have employed Binary Relevance & OneVsRest approach for model building.

3. Key Metrics for Success in Solving Problem Under Consideration

- **The evaluation measures for single-label are usually different than for multi-label.** Here in single-label classification we use simple metrics such as precision, recall, accuracy, etc, say, in single-label classification, accuracy is just:

$$\frac{1}{N}\sum_{i=1}^{N}\mathcal{I}[\hat{y}^{(i)} = y^{(i)}]$$

Fig. Accuracy in single-label classification

- **In multi-label classification, a misclassification is no longer a hard wrong or right.** A prediction containing a subset of the actual classes should be considered better than a prediction that contains none of them, i.e., **predicting two of the three labels correctly this is betterthan predicting no labels at all.**

Micro-averaging & Macro-averaging (Label based measures):

- To measure a multi-class classifier, we have to average out the classes somehow. There are two different methods of doing this called *micro-averaging and macro-averaging*.
- In *micro-averaging* all TPs, TNs, FPs and FNs for each class are summed up and then the average is taken.

- Microaveraging Precision $Prc^{micro}(D) = \frac{\sum_{c_i \in \mathcal{C}} TPs(c_i)}{\sum_{c_i \in \mathcal{C}} TPs(c_i) + FPs(c_i)}$
- Microaveraging Recall $Rcl^{micro}(D) = \frac{\sum_{c_i \in \mathcal{C}} TPs(c_i)}{\sum_{c_i \in \mathcal{C}} TPs(c_i) + FNs(c_i)}$

Fig. Micro-Averaging

- In *micro-averaging* method, you sum up the individual true positives, false positives, and false negatives of the system for different sets and the apply them. And the micro-average F1-Score will be simply the harmonic mean of above two equations.
- *Macro-averaging* is straight forward. We just take the average of the precision and recall of the system on different sets.

- Macroaveraging Precision $Prc^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Prc(D, c_i)}{|\mathcal{C}|}$
- Macroaveraging Recall $Rec^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Rcl(D, c_i)}{|\mathcal{C}|}$

Fig. Macro-Averaging

- *Macro-averaging* method can be used when you want to know how the system performs overall across the sets of data. You should not come up with any specific decision with this average. On the other hand, *micro-averaging* can be a useful measure when your dataset varies in size.

Hamming-Loss (Example based measure):

- In simplest of terms, *Hamming-Loss* is the fraction of labels that are incorrectly predicted, i.e., the fraction of the wrong labels to the total number of labels.

$$\frac{1}{|N| \cdot |L|} \sum_{i=1}^{|N|} \sum_{j=1}^{|L|} \text{xor}(y_{i,j}, z_{i,j}), \text{ where } y_{i,j} \text{ is the target and } z_{i,j} \text{ is the prediction.}$$

Fig. Hamming-Loss

Exact Match Ratio (Subset accuracy):

- It is the strictest metric, indicating the percentage of samples that have all their labels classified correctly.

$$ExactMatchRatio, MR = \frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i)$$

Fig. Exact Match Ratio

- The disadvantage of this measure is that multi-class classification problems have a chance of being partially correct, but here we ignore those partially correct matches.
- There is a function in *scikit-learn* which implements subset accuracy, called as accuracy_score.

4. Run And Evaluate Selected Models

```
# 3. Training and Testing Model on our train dataset

# Creating a function to train and test model
def build_models(models,x,y,test_size=0.33,random_state=42):
    # spliting train test data using train_test_split
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=test_size,random_state=random_state)

    # training models using BinaryRelevance of problem transform
    for i in tqdm.tqdm(models,desc="Building Models"):
        start_time = timeit.default_timer()

        sys.stdout.write("\n=========================================================================================\n")
        sys.stdout.write(f"Current Model in Progress: {i} ")
        sys.stdout.write("\n=========================================================================================\n")

        br_clf = BinaryRelevance(classifier=models[i]["name"],require_dense=[True,True])
        print("Training: ",br_clf)
        br_clf.fit(x_train,y_train)

        print("Testing: ")
        predict_y = br_clf.predict(x_test)

        ham_loss = hamming_loss(y_test,predict_y)
        sys.stdout.write(f"\n\tHamming Loss  : {ham_loss}")

        ac_score = accuracy_score(y_test,predict_y)
        sys.stdout.write(f"\n\tAccuracy Score: {ac_score}")

        cl_report = classification_report(y_test,predict_y)
        sys.stdout.write(f"\n{cl_report}")

        end_time = timeit.default_timer()
        sys.stdout.write(f"Completed in [{end_time-start_time} sec.]")

        models[i]["trained"] = br_clf
        models[i]["hamming_loss"] = ham_loss
        models[i]["accuracy_score"] = ac_score
        models[i]["classification_report"] = cl_report
        models[i]["predict_y"] = predict_y
        models[i]["time_taken"] = end_time - start_time

        sys.stdout.write("\n=========================================================================================\n")

    models["x_train"] = x_train
    models["y_train"] = y_train
    models["x_test"] = x_test
    models["y_test"] = y_test

    return models
```

The function is code as shown in above pic to build model using earlier mention algorithms. The evaluation matrix for these algorithms are shown below:

1. Logistic Regression

```
================================================================================================
Current Model in Progress: Logistic Regression
================================================================================================
Training:  BinaryRelevance(classifier=LogisticRegression(), require_dense=[True, True])
Testing:

        Hamming Loss  : 0.022066084314470186
        Accuracy Score: 0.9123433345993164
              precision    recall  f1-score   support

           0       0.92      0.51      0.66      1281
           1       0.61      0.18      0.28       150
           2       0.95      0.54      0.69       724
           3       0.00      0.00      0.00        44
           4       0.81      0.45      0.58       650
           5       0.88      0.13      0.22       109

   micro avg       0.89      0.47      0.61      2958
   macro avg       0.70      0.30      0.40      2958
weighted avg       0.87      0.47      0.60      2958
 samples avg       0.05      0.04      0.04      2958
Completed in [48.712080600000036 sec.]
================================================================================================
```

2. Random Forest Classifier

```
================================================================================================
Current Model in Progress: Random Forest Classifier
================================================================================================
Training:  BinaryRelevance(classifier=RandomForestClassifier(), require_dense=[True, True])
Testing:

        Hamming Loss  : 0.021914166350170 91
        Accuracy Score: 0.9063425750094949
              precision    recall  f1-score   support

           0       0.81      0.60      0.69      1281
           1       0.61      0.07      0.13       150
           2       0.87      0.70      0.78       724
           3       0.00      0.00      0.00        44
           4       0.69      0.53      0.60       650
           5       0.80      0.15      0.25       109

   micro avg       0.79      0.56      0.66      2958
   macro avg       0.63      0.34      0.41      2958
weighted avg       0.78      0.56      0.64      2958
 samples avg       0.05      0.05      0.05      2958
Completed in [1184.5038663999999 sec.]
================================================================================================
```

3. Support Vector Machine

```
==========================================================================================
Current Model in Progress: Support Vector Classifier
==========================================================================================
Training:  BinaryRelevance(classifier=LinearSVC(max_iter=3000), require_dense=[True, True])
Testing:

        Hamming Loss  : 0.020952019242942144
        Accuracy Score: 0.9115077857956704
              precision    recall  f1-score   support

           0       0.85      0.60      0.71      1281
           1       0.49      0.16      0.24       150
           2       0.90      0.65      0.75       724
           3       0.50      0.18      0.27        44
           4       0.75      0.53      0.62       650
           5       0.80      0.32      0.46       109

   micro avg       0.82      0.56      0.67      2958
   macro avg       0.71      0.41      0.51      2958
weighted avg       0.81      0.56      0.66      2958
 samples avg       0.06      0.05      0.05      2958
Completed in [4.937557199999901 sec.]
==========================================================================================
```

4. Ada Boost Classifier

```
==========================================================================================
Current Model in Progress: Ada Boost Classifier
==========================================================================================
Training:  BinaryRelevance(classifier=AdaBoostClassifier(), require_dense=[True, True])
Testing:

        Hamming Loss  : 0.023446005823521965
        Accuracy Score: 0.9057349031522978
              precision    recall  f1-score   support

           0       0.82      0.53      0.65      1281
           1       0.51      0.23      0.32       150
           2       0.90      0.60      0.72       724
           3       0.53      0.18      0.27        44
           4       0.71      0.44      0.54       650
           5       0.60      0.28      0.39       109

   micro avg       0.80      0.50      0.61      2958
   macro avg       0.68      0.38      0.48      2958
weighted avg       0.79      0.50      0.61      2958
 samples avg       0.05      0.04      0.04      2958
Completed in [546.8039282 sec.]
==========================================================================================
```

From the above model comparison, it is clear that Linear Support Vector Classifier performs better with Accuracy Score: 91.1508 % and Hamming Loss: 2.0953% than the other classification models. Hyperparameter tuning is applied over Linear Support Vector Classifier and used it as final model.

```
from sklearn.model_selection import  GridSearchCV

fmod_param = {'estimator__penalty' : ['l1', 'l2'],
              'estimator__loss' : ['hinge', 'squared_hinge'],
              'estimator__multi_class' : ['ovr', 'crammer_singer'],
              'estimator__random_state' : [42, 72, 111] }
#SVC = BinaryRelevance(classifier=LinearSVC(),require_dense=[True,True])
SVC = OneVsRestClassifier(LinearSVC())
GSCV = GridSearchCV(SVC, fmod_param, cv=3,verbose = 10)
x_train,x_test,y_train,y_test = train_test_split(X[:half,:], Y[:half,:], test_size=0.30, random_state=42)
GSCV.fit(x_train,y_train)
GSCV.best_params_
```

```
[CV 3/3; 23/24] END estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator__r
andom_state=72;, score=0.912 total time=   6.2s
[CV 1/3; 24/24] START estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator_
_random_state=111
[CV 1/3; 24/24] END estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator__r
andom_state=111;, score=0.914 total time=  14.5s
[CV 2/3; 24/24] START estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator_
_random_state=111
[CV 2/3; 24/24] END estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator__r
andom_state=111;, score=0.915 total time=  10.8s
[CV 3/3; 24/24] START estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator_
_random_state=111
[CV 3/3; 24/24] END estimator__loss=squared_hinge, estimator__multi_class=crammer_singer, estimator__penalty=l2, estimator__r
andom_state=111;, score=0.912 total time=   8.8s

{'estimator__loss': 'hinge',
 'estimator__multi_class': 'ovr',
 'estimator__penalty': 'l2',
 'estimator__random_state': 42}
```

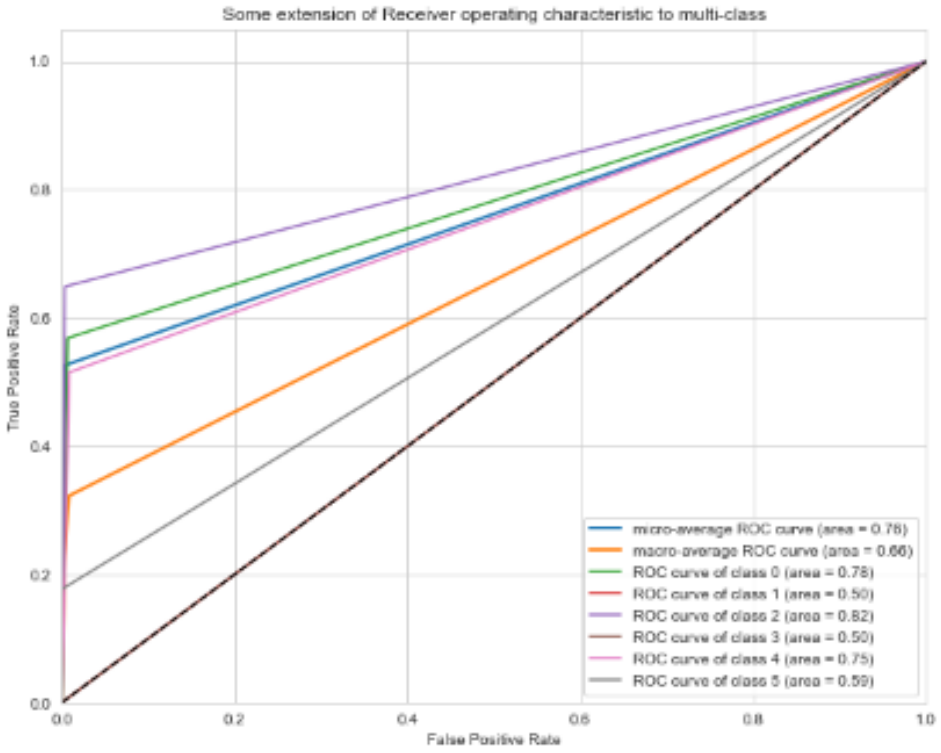
Final Model is Build using best params obtain in hyper parameter tuning.The corresponding final model is shown below:

Final Model

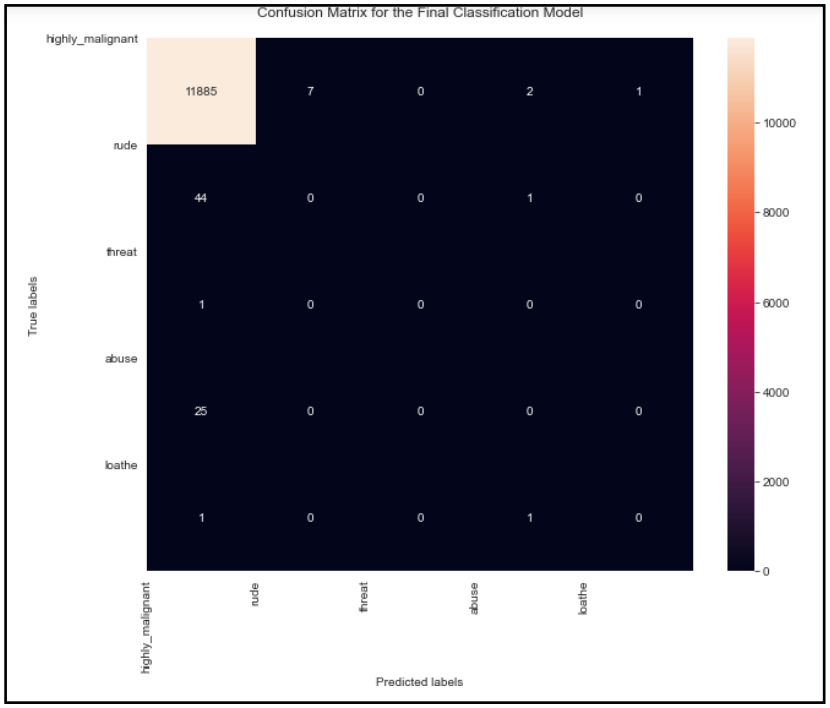
```
Final_Model = OneVsRestClassifier(LinearSVC(loss='hinge', multi_class='ovr', penalty='l2', random_state=42))
Classifier = Final_Model.fit(x_train, y_train)
fmod_pred = Final_Model.predict(x_test)
fmod_acc = (accuracy_score(y_test, fmod_pred))*100
print("Accuracy score for the Best Model is:", fmod_acc)
h_loss = hamming_loss(y_test,fmod_pred)*100
print("Hamming loss for the Best Model is:", h_loss)
```

```
Accuracy score for the Best Model is: 91.26002673796792
Hamming loss for the Best Model is: 2.0819407308377897
```

AOC ROC of Final Model:

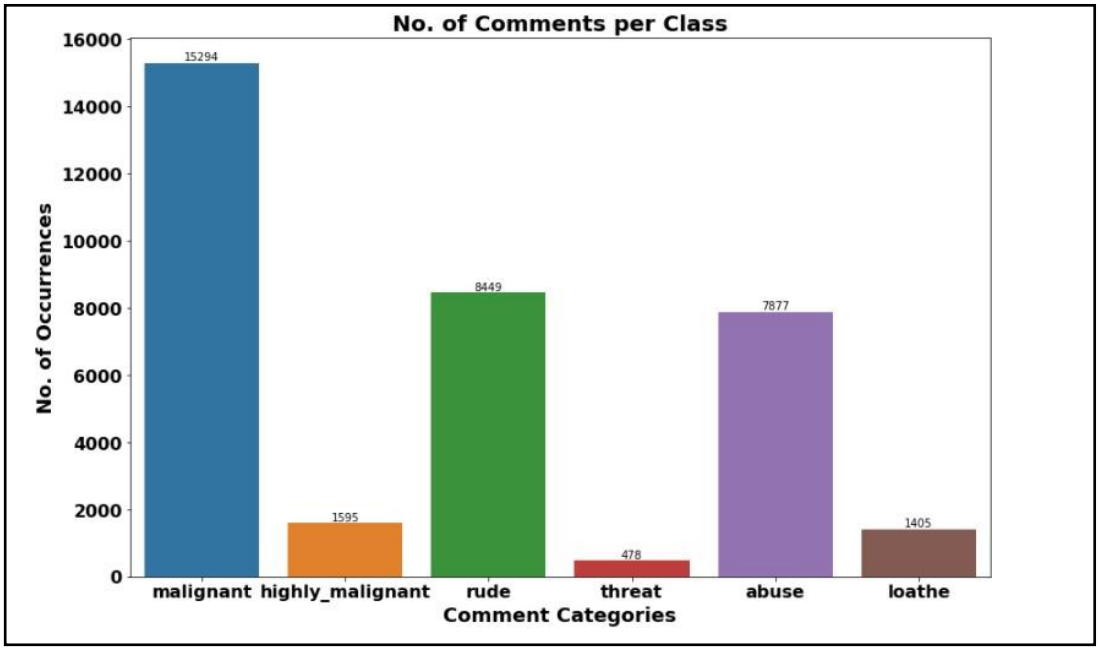


Confusion Matrix of Final Model:



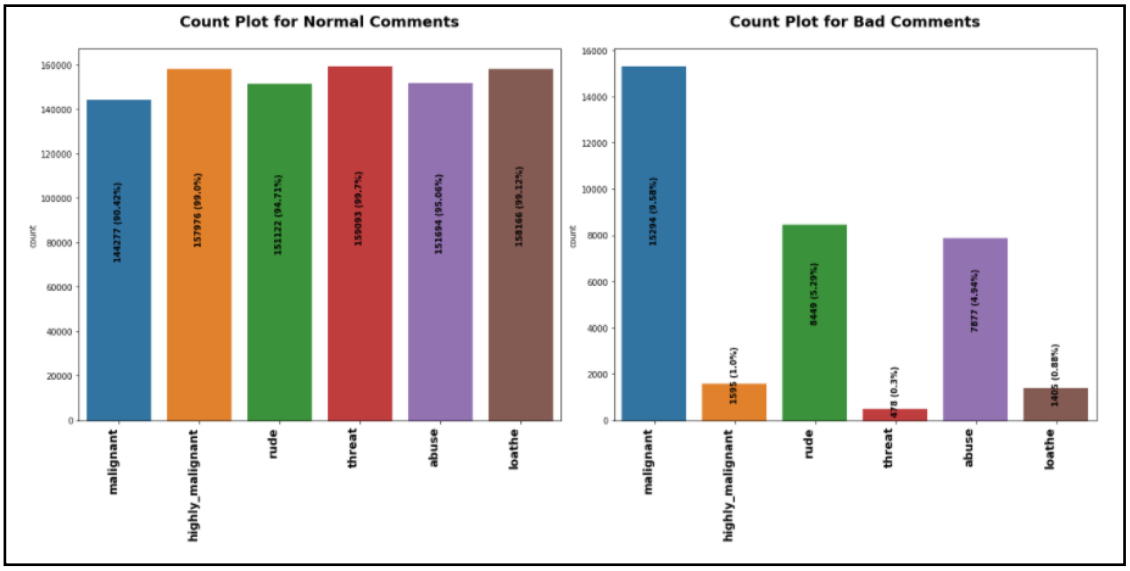
5.Visualizations

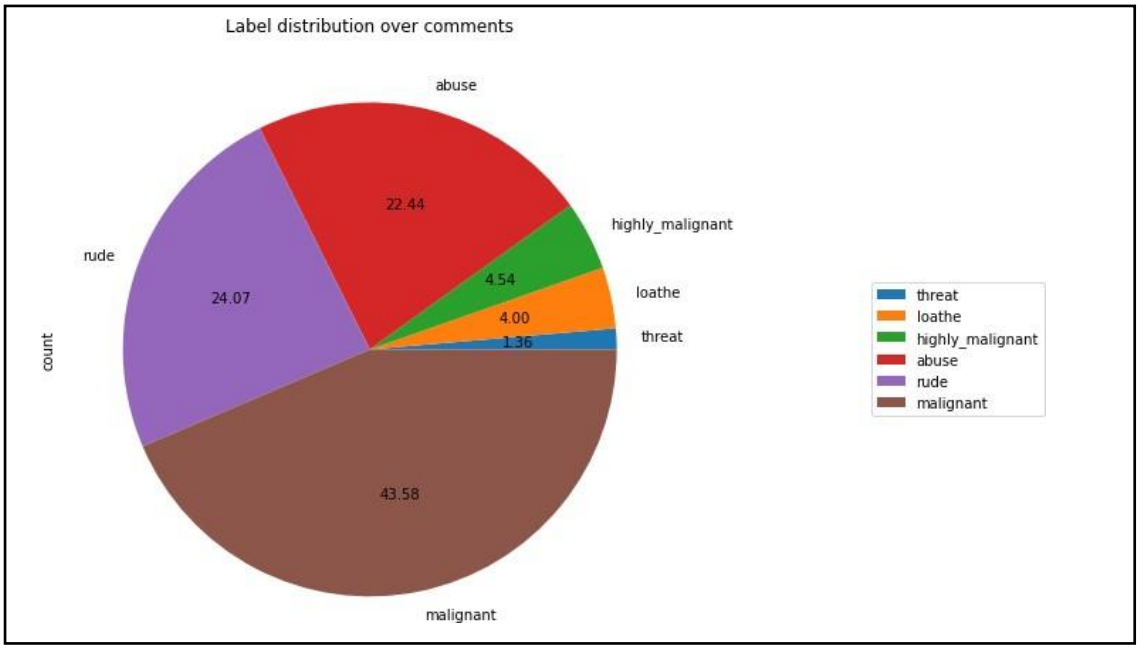
There are 6 different labels in dataset. Let's explore total no of comments per class.



Observation:

- Out of total Negative comments the maximum negative comments come with Malignant in nature followed by rude categories.
- Around 90% comments are Good/Neutral in nature while rest 10% comments are Negative in nature.
- Very few comments come with threatening nature.

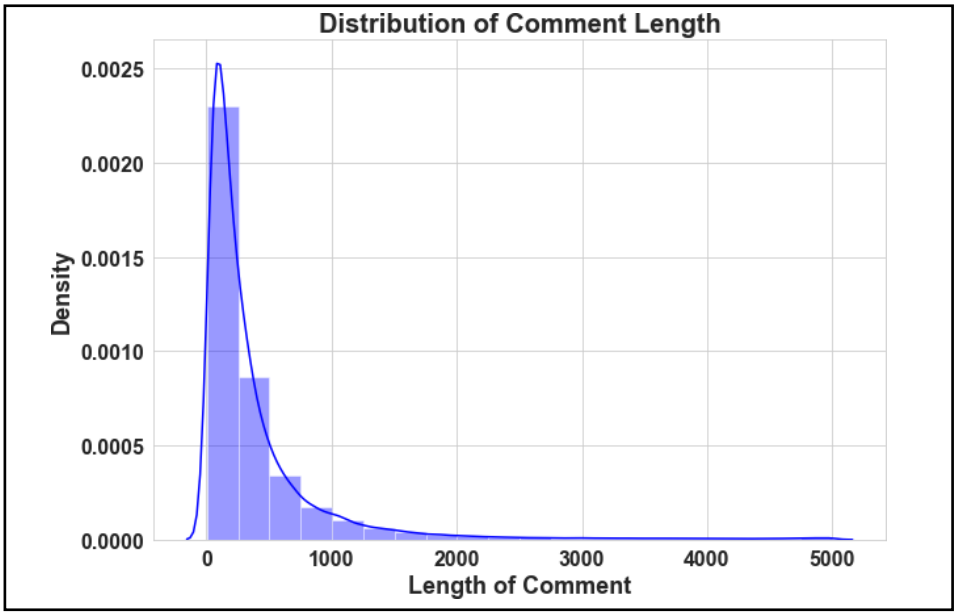




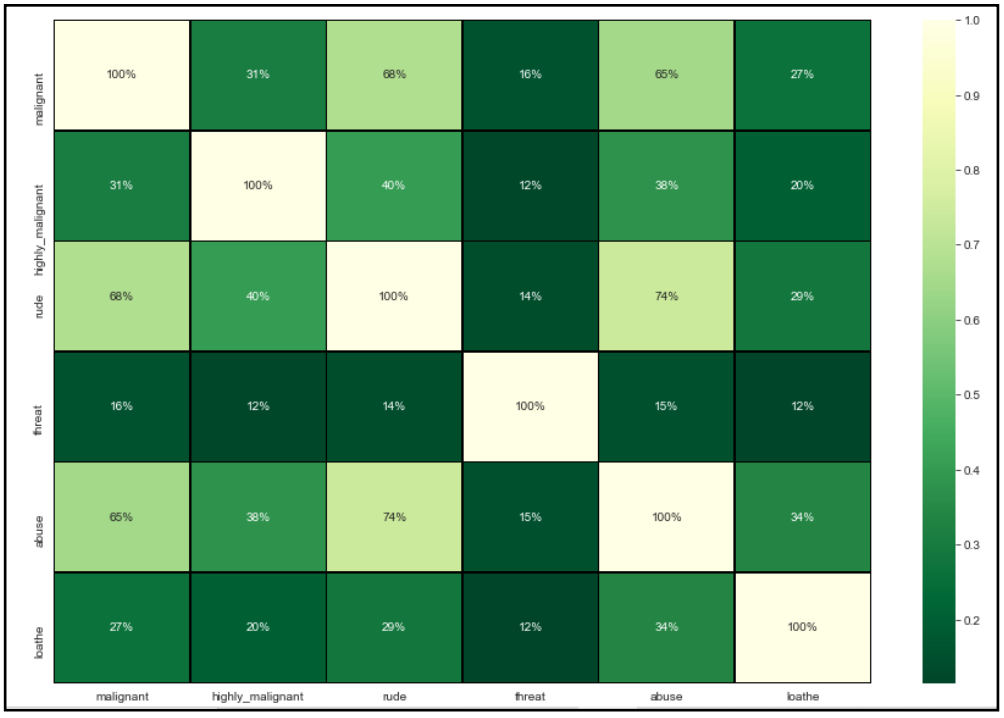
Observation:

- Around 90% comments are Good/Neutral in nature while rest 10% comments are Negative in nature.

- Out of total negative comments around 43.58% are malignant in nature followed by 24.07% are rude comments.



- Above is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words. Majority of the comments are of length 500, where maximum length is 5000 and minimum length is 5. Median length being 250.



Observation:

- The highest positive correlation is seen in between fields 'rude' and 'abuse'.
- Attribute 'threat' is negatively correlated with each and every other feature of this training dataset.
- Almost all variable is correlated with each other negatively.

Chap 4. Conclusion

1. Key Findings and Conclusions of the Study

Algorithm	Accuracy Score	Recall (Micro)	Precision (Micro)	F1 Score (Micro)	Humming Loss
Logistics Regression	0.9123	0.89	0.94	0.61	0.02206
Random Forest Classifier (RFC)	0.9074	0.56	0.79	0.66	0.02191
Support Vector Classifier	0.9115	0.56	0.82	0.67	0.02952
Ada Boost Classifier	0.9057	0.50	0.80	0.61	0.9057

- Linear Support Vector Classifier performs better with Accuracy Score: 91.15077857956704 % and Hamming Loss: 2.0952019242942144
  % than the other classification models. Hyperparameter Tuning is performed over this model to enhance accuracy.
- Final Model is giving us Accuracy score of 91.26% which is slightly improved compare to earlier Accuracy score of 91.15%.
- SVM classifier is fastest algorithm compare to others.

2. Learning Outcomes of the Study in respect of Data Science

- In this project we were able to learn various Natural language processing techniques like lemmatization, stemming, removal of Stop words.
- This is my first multi-label classification-based NLP project.

3. Limitations of this work and Scope for Future Work

- The Maximum feature used while vectorization is 2000. Employing more feature in vectorization lead to more accurate model which I not able to employed due computational resources.

- Data is imbalanced in nature but due to computational limitation we have not employed balancing techniques here.
- Deep learning CNN, ANN can be employed to create more accurate model.
- Every effort has been put on it for perfection but nothing is perfect and this project is of no exception. There are certain areas which can be enhanced. Comment detection is an emerging research area with few public datasets. So, a lot of works need to be done on this field.