FLIP ROBO

PROJECT REPORT ON:

“Malignant Comment Classifier”

SUBMITTED BY

D Krishna Teja

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 Ms.Gulshana Chaudhary (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 my academic team "Data trained" who are the reason behind what I am today. Last but not least my parents who have been my backbone in every step of my life. And also thank you for many other persons who has helped me directly or indirectly to complete the project.

Contents:

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 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 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 forum and social media platforms have provided individuals with the means to put forward their thoughts and freely express their opinion on various issues and incident. In some cases, these online comments contain explicit language which may hurt the readers. Comments containing explicit language can be classified into myriad categories such as Malignant, Highly Malignant, Rude, Threat, Abuse and Loathe. The threat of abuse and harassment means that 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.

1.3 Review of Literature

The purpose of the literature review is to:

1. Identify the foul words or foul statements that are being used.

2. Stop the people from using these foul languages in online public forum.

To solve this problem, we are now building a model using our machine learning that identifies all the foul language and foul words, using which the online platforms like social media principally stops these mob using the foul language in an online community or even block them or block them from using this foul language.

I have used different Classification algorithms and shortlisted the best on basis of the metrics of performance and I have chosen one algorithm and build a model in that algorithm.

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.

1.4 Motivation for the Problem Undertaken

Now a days social media users are increasing continuously so regularization on social media platform is very necessary. Many users are abuse or harass from social media, and teenager are having bad impact of this because they may face threat call or massages social media due to this, they take unwanted or unnecessary step. For Avoiding this, if we install a machine whose filter out the comment. So, we will be decreasing the thus threat or unwanted activities from social media platforms. One of the first lessons we learn as children is that the louder you scream and the bigger of a tantrum you throw, you more you get your way. Part of growing up and maturing into an adult and functioning member of society is learning how to use language and reasoning skills to communicate our beliefs and respectfully disagree with others, using evidence and persuasiveness to try and bring them over to our way of thinking.

2.Analytical Problem Framing

2.1 Mathematical/ Analytical Modelling of the Problem

The libraries/dependencies imported for this project are shown below:

```
In [1]: #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')
```

Here in this project, we have been provided with two datasets namely train and test CSV files. I will build a machine learning model using train dataset. And using this model we will make predictions for our test dataset.

2.2 Data Sources and their formats

We have been provided with two datasets namely train and test CSV filers. Train datasets contains 159571 rows and 8 columns.

```
In [2]: # Importing dataset excel file using pandas.
        df=pd.read_csv(r"C:\Users\Teja\Downloads\Malignant-Comments-Classifier-Project\train.csv")
```

```
In [3]: 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
```

Out[3]:

	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 set consist of 159571 rows and 8 columns. In this malignant, highly_malignant,rude,threat, abuse and loathe are our target variables having binary class of Yes (1) & No (2).

Test datasets contains 153164 rows and 8 columns.

- **Loading Test Dataset**

```
In [5]: # Importing dataset excel file using pandas.
        dft=pd.read_csv(r"C:\Users\Teja\Downloads\Malignant-Comments-Classifier-Project\test.csv")
```

```
In [6]: print('No. of Rows :',dft.shape[0])
        print('No. of Columns :',dft.shape[1])
        pd.set_option('display.max_columns',None) # This will enable us to see truncated columns
        dft.head()

        No. of Rows : 153164
        No. of Columns : 2
```

Out[6]:

	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.

2.3 Data Pre-processing Done

- First step I have imported required libraries and I have imported the dataset which was in csv format.
- Then I did all the statistical analysis like checking shape, nunique, value counts, info etc.
- I found that, only one features are used to make prediction and here are 6 categories to predict.
- Checked for any missing values in the datasets.
- Then doing some EDA and Building Models.

2.4 Hardware, Software and Tool Used

Hardware Used:

Processor – Intel core i5

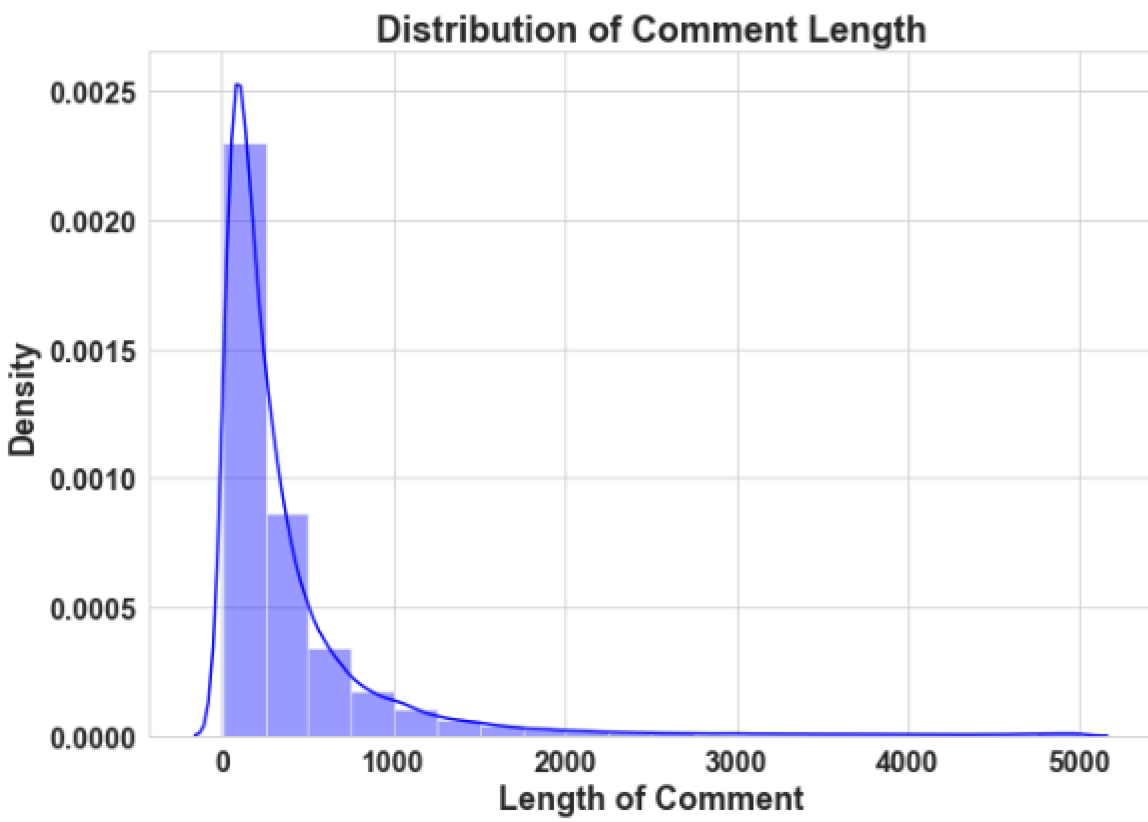
Physical Memory – 16 GB

Software Used:

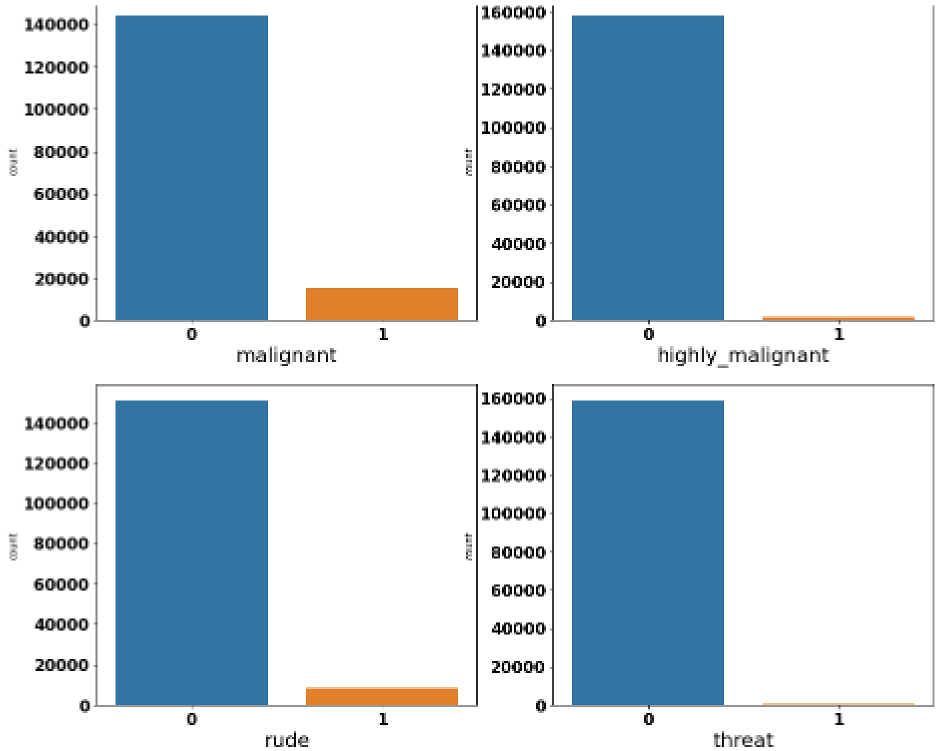
- Windows 10 Operating System
- Anaconda Package and Environment Manager
- Jupyter Notebook
- Python Libraries used: In Which Pandas, Seaborn, Matplotlib, Numpy and Scipy
- sklearn for Modelling Machine learning algorithms, Data Encoding, Evaluation metrics, Data Transformation, Data Scaling, Component analysis, Feature selection etc.

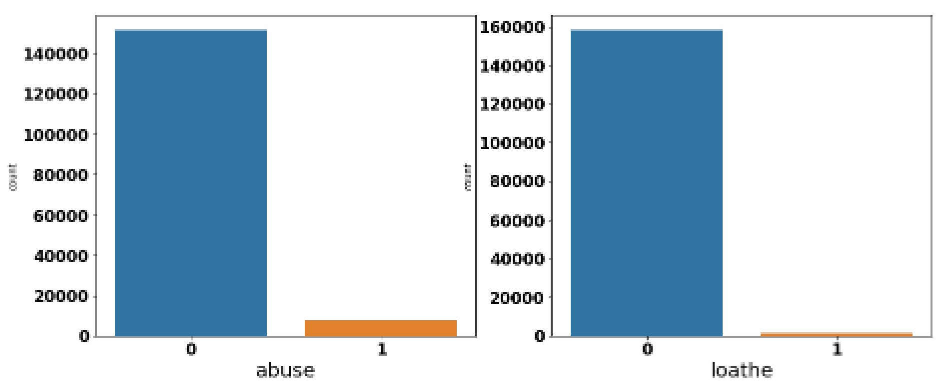
3.Data Analysis and Visualization

3.1 Univariate Visualization



- We can see, up to 2000 words comment test are used. It seems to be people are used so many malignant words while commenting.
- But there is also more 4000 words but in less numbers are also used malignant words to commenting.



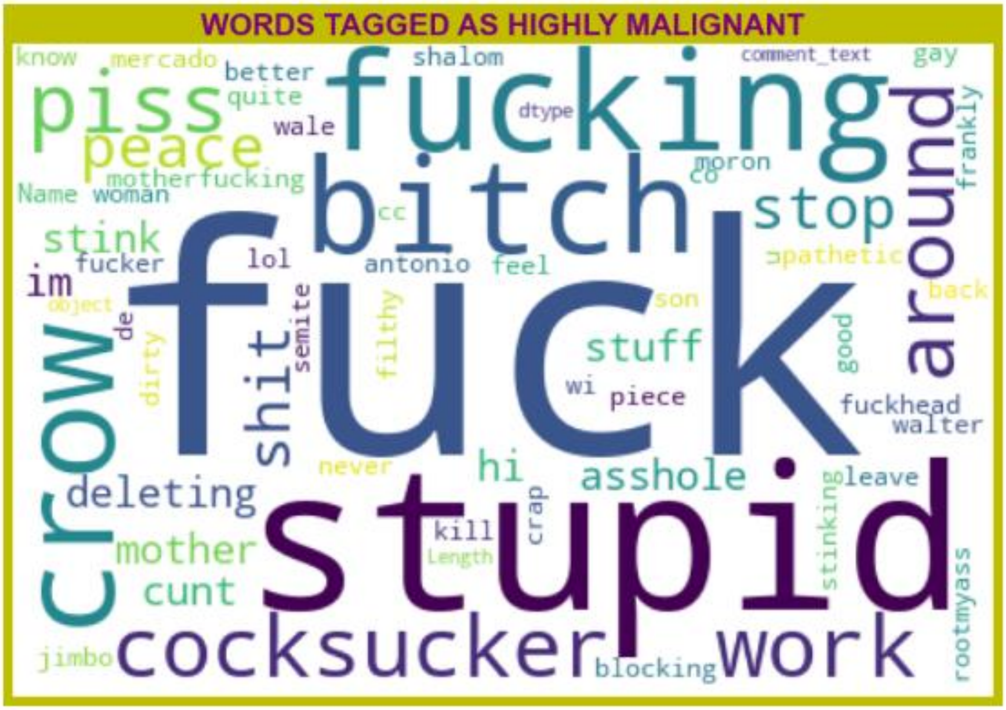


- We can see, in malignant categories of comment there are having less malignant comment. 1 has low count than 0 it means that0 may be normal comment 1 may be malignant.
- In Highly malignant 1 has less than 0 it meant that there is less highly malignant word using by users. But their magnitude is high because it may cause the depression to others users.

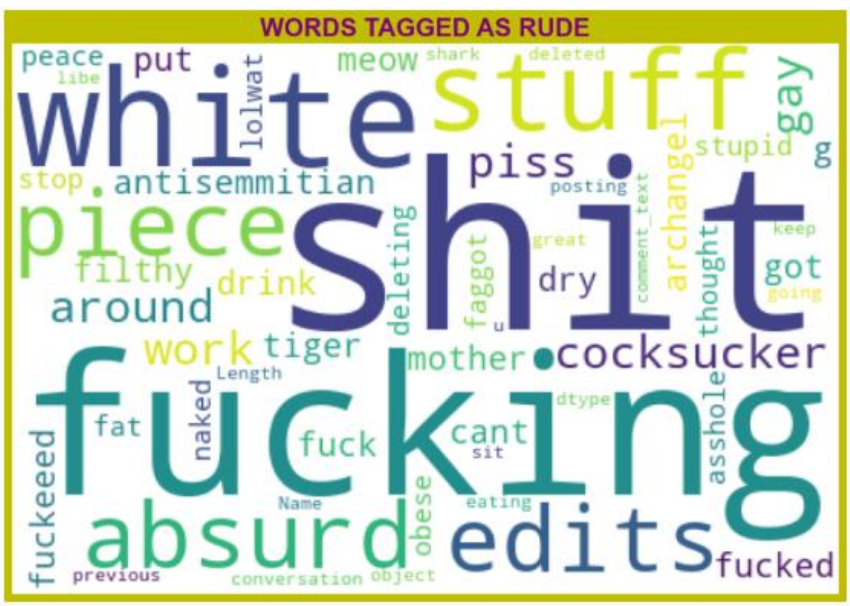
- In loathe, there is also low percentage of 1 category.
- In rude categories, rude comment is high than others categories so it may cause another user.
- Rude and Abuse is having almost same quantity of comment but their magnitude or impact will be different. Rude has low impact than abuse comment.
- We can see there is also threat comment but their ration very small but it may have very large impact.
- 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.

3.2 Word Cloud Visualization

- From wordcloud of malignant comments, it is clear that it mostly consists of words like edits, hey, white, fucking, gay, cocksucker, work, think, taliban etc.

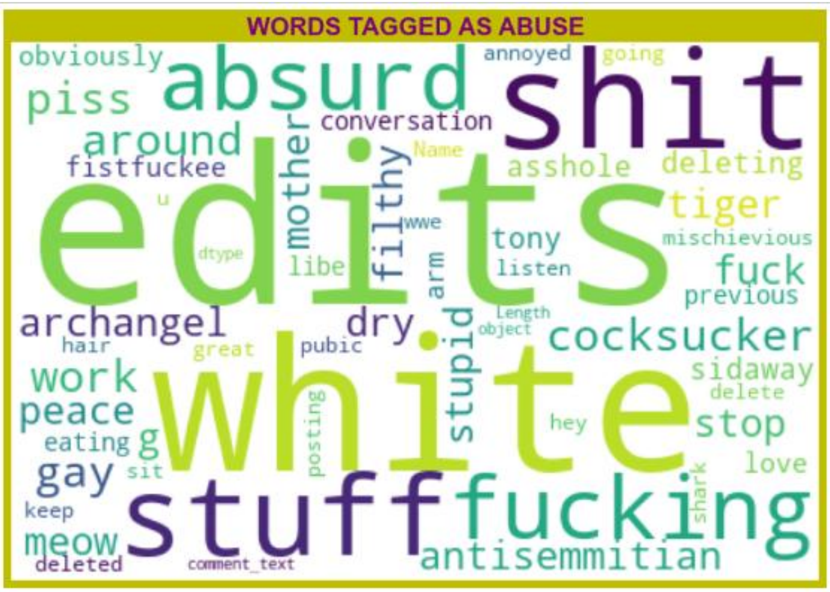


- *From word cloud of* Highly malignant comments*, it is clear that it mostly consists of words like fuck, stupid, fucking, bitch, crow, shit, cocksucker etc.*

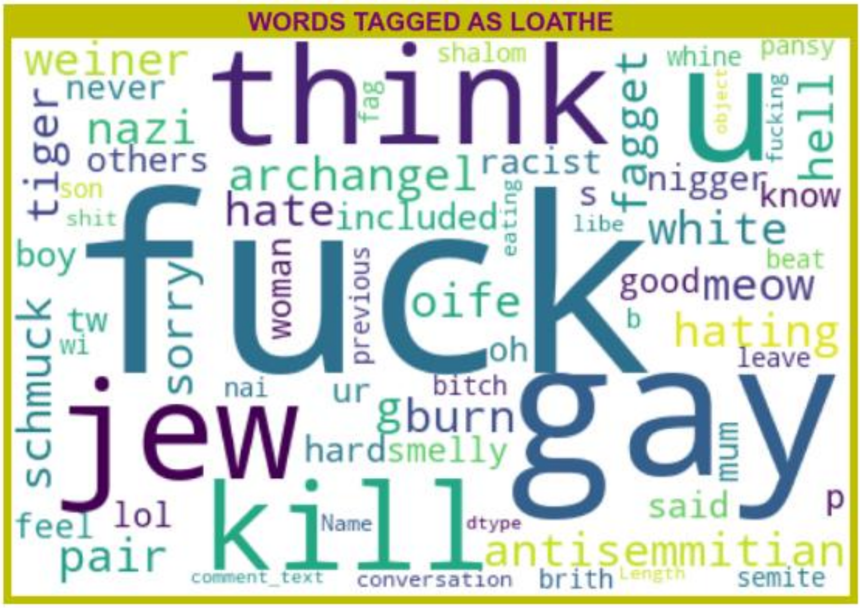


- *From word cloud of* Rude comments*, it is clear that it mostly consists of words like fucking, shit, white, piece, edits, stuff, absurd etc.*

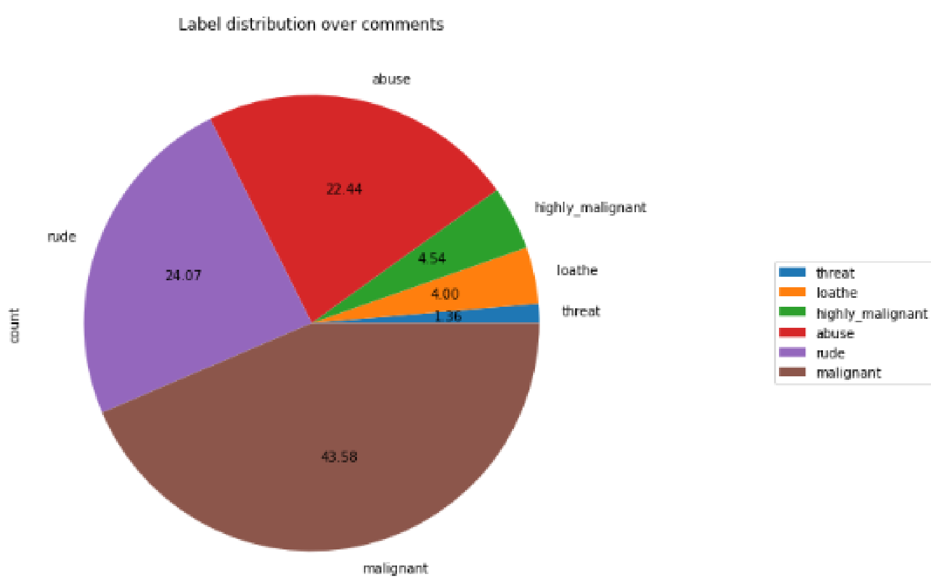
- *From word cloud of* Threat comments*, it is clear that it mostly consists of words like fuck, suck, Bitch, die, stupid, etc.*



- *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 wordcloud of Loathe comments, it is clear that it mostly consists of words like fuck,gay, kill, think, Jew, u etc.



- We can see, malignant is having maximum comment than others categories followed by rude.
- Threat categories comment is having low count but it has high impact than others.
- Highly malignant comments are having high impact on user whose face such problem.

- 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 are correlated with each other negatively.

4. **Models Development and Evaluation**

```
In [26]: #Importing Required Libraries
import nltk
import re
import string
from nltk.corpus import stopwords
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [27]: #Defining the stop words
stop_words = stopwords.words('english')

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

```
In [28]: #Replacing '\n' in comment_text
df['comment_text'] = df['comment_text'].replace('\n',' ')
```

```
In [29]: #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)
```

Separating independent and dependent variables

- 1. Vectorizer & Spliting Train dataset

In [42]:
```
# Converting the features into number vectors
tf_vec = TfidfVectorizer(max_features = 2000, stop_words='english')
```

In [43]:
```
# 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()
```

In [44]:
```
output_labels= df.columns[1:7]
```

In [45]:
```
# 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)
```

- 2. Vectorizer & Spliting Train dataset

In [46]:
```
# Doing the above process for test data
test_vec = tf_vec.fit_transform(dft['comment_text'])
test_vec
```

Out[46]:
```
<153164x2000 sparse matrix of type '<class 'numpy.float64'>'
	with 2138199 stored elements in Compressed Sparse Row format>
```

In [47]:
```
test_vec.shape
```

Out[47]:
```
(153164, 2000)
```

Machine Learning Model Building

In [48]:
```
#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
```

In [49]:
```
import timeit, sys
import tqdm.notebook as tqdm
```

```
In [50]: # 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
```

```
Building Models:   0%|          | 0/4 [00:00<?, ?it/s]

=====================================================================================
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 [15.706447800000035 sec.]
=====================================================================================
```

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

        Hamming Loss  : 0.02195214584124573
        Accuracy Score: 0.9076338777060388
              precision    recall  f1-score   support

           0       0.81      0.61      0.69      1281
           1       0.55      0.07      0.13       150
           2       0.87      0.69      0.77       724
           3       0.00      0.00      0.00        44
           4       0.69      0.53      0.60       650
           5       0.85      0.16      0.26       109

   micro avg       0.80      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 [967.2329419 sec.]
==================================================================================
```

```
==================================================================================
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 [4.379729900000029 sec.]
==================================================================================
```

```
==================================================================================
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 [596.2665637 sec.]
==================================================================================
```

4.2 Interpretation of the results

From the above model comparisons it is clear that Linear Support Vector Classifier performs better with Accuracy Score: 91.15077857956704 % and Hamming Loss: 2.0952019242942144 % than the other classification models.

4.3 Hyperparameter Tuning

```
In [52]: from sklearn.model_selection import  GridSearchCV
```

```
In [53]: 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_
```

```
Out[53]: {'estimator__loss': 'hinge',
          'estimator__multi_class': 'ovr',
          'estimator__penalty': 'l2',
          'estimator__random_state': 42}
```

Final Model

```
In [54]: 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
```

Final Model is giving us Accuracy score of 91.26% which is slightly improved compare to earlier Accuracy score of 91.15%.

5. Conclusions

5.1 Key Finding and Conclusions

- Linear Support Vector Classifier performs better with Accuracy Score: 91.15077857956704 % and Hamming Loss: 2.0952019242942144 % than the other classification models.
- Final Model (Hyperparameter Tuning) 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.

5.2 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.