FLIP ROBO

NAME OF THE PROJECT

Malignant Comment Classifier

Submitted by:

Shilpi Mohanty

ACKNOWLEDGMENT

I am highly indebted to Flip Robo Technologies Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the Project and also for their support in completing the project.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to Mr. Mohd. Kashif, SME for giving the dataset and clear instructions to perform the complete case study process and guided and advised me from time to time.

I perceive as this opportunity as a learning experience in my career development. I will strive to use gained skills and knowledge in the best possible way, and try to work on the improvement, in order to attain desired career objectives.

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 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.

- ## Conceptual Background of the Domain Problem

  Information and communication technology has changed rapidly over the past 20 years, with a key development being the emergence of social media.

  Platforms that aggregate user content are the foundation of knowledge sharing on the Internet. Blogs, forums, discussion boards, and, of course, Wikipedia. But the catch is that not all people on the Internet are interested in participating nicely, and some see it as an avenue to vent their rage, insecurity, and prejudices.

  In most of the online conversation platforms, social media users often face abuse, harassment, and insults from other users. Accordingly, a jargon word has been coined recently to address such behaviours as “cyberbullying”. The problem with this is that people will frequently write things they shouldn’t, and to maintain a positive community this toxic content and the users posting it need to be removed quickly. But they don’t have the resources to hire full-time moderators to review every comment. Due to which, many users stop expressing their ideas and opinions. Platforms struggle to facilitate conversations effectively.

  To solve this problem, we are now building a model 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.

- ## Review of Literature

There is lot of information being delivered every time on social media sites. There is increase in hate speech that both may promotes violence towards. There was survey conducted asking American adult about problem of online harassment or bullying. Roughly four-in-ten Americans have personally experienced online harassment. 62% percent of participant in study considered it as major problem whereas 33% considered as minor problem. A total of 95% called it problem and 35% agreed for online companies to build better policies and tools for their platforms. Due to negativity, civilized conversations via social media are not present since hateful remarks are limiting individual to communicate and to have contradicting feelings.

There were endeavours by people to build the online wellbeing by moderating websites through crowd- sourcing schemes and remark criticizing, much of the time these procedures neglect to recognize the toxicity. Along these lines, we need to track down a potential method that can recognize the online toxicity of client content successfully.

As Computer understands binary information and in real world we have information in different structures for example pictures or text. So, we need to change over the information of real world into binary for legitimate processing through the computer. In this paper, they have utilized this changed over information and apply Machine learning strategies to arrange online remarks.

Nguyen and Nguyen made model consisting of 2 components Deep Learning Classifier and Tweet Processor. Tweet Processor is used for applying semantic rules and preprocessing on datasets to capture important information. They used character-level embeddings to increase information for word-level embedding. They then used DeepCNN for character level embeddings. After that a Bidirectional Long Short-Term Memory network (BiLSTM) produces a sentence-wide feature representation from the global fixed size feature and word- level embedding. Their model produce an accuracy of 86.63% on Standford Twitter Sentiment Corpus.

Liang-Chih Yu et al. proposes a word vector refinement model. This refinement can be applied to any of already trained word vectors. Based

on semicolon lexicon, their model gives high rank to sentimentaly similar neighbour and vice versa. Their experimental results show that their proposed method can perform better for various neural networks. It also have better performance for both sentimental embedding and conventional word embeddings.

A method was introduced by Wulczyn et al. develop method to analyze personal attacks. Theygenerated over 63M machine labeled and 100k human labeled comments. They found that attacks on Wikipedia are not limited to a set of users.

Hossein Hosseini et al. [15] apply the attack on the Perspective toxic comment detection website. This website gives toxic score to any phrase. They tried to modify a toxic phrase having same meaning so that model will give it very low toxic scores. This existence is harmful for toxic detection system.

In another methodology, Convolutional Neural Networks (CNN) was utilized in text characterization over online substance [16], with no information on syntactic or semantic language.

Y. Chen et al. [17] propose the Lexical Syntactic Feature (LSF) architecture to distinguish offensive content and recognize likely offensive clients in web-based media. Their experiments shows that LSF algorithms for sentence and user offensiveness outperformed traditional learning-based methods. Their LSF can adapt to various writing styles of English language and can tolerate informal and misspelled content.

Jigsaw and Googles Counter Abuse Technology team introduce project named Perspective. It uses machine learning models to identify abusive comments. The models score a phrase based on the perceived impact the text may have in a conversation and have capability to classifying comments.

- ## Motivation for the Problem Undertaken

  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.

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

Analytical Problem Framing

• Mathematical/ Analytical Modelling of the Problem

There are two different datasets i.e. train and test dataset provided to us. The training data consist of both dependent and independent variables. As it is a multiclass-label problem it has 6 independent/target variables. Here the target variables are named as “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.

Clearly it is a binary classification problem as the target columns giving binary outputs and all independent variables have text so it is clear that it is a supervised machine learning problem where we can use the techniques of NLP and classification-based algorithms of Machine Learning. Here we will use NLP techniques like word tokenization, lemmatization, stemming and tfidf vectorizer then those processed data will be used to create best model using various classification based supervised ML algorithms like Logistic Regression, Multinomial NB, LGBM Classifier, LinearSVC, Passive aggressive Classifier.

• Data Sources and their formats

Data set provided by Flip Robo was in the format of CSV (Comma Separated Values). 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.

Thus, data consists of one input feature, the string data for the comments, and six labels for different categories of toxic comments: Malignant, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

The data set includes:

Columns	Description
Malignant:	It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
Highly Malignant:	It denotes comments that are highly malignant and hurtful.
Rude:	It denotes comments that are very rude and offensive.
Threat:	It contains indication of the comments that are giving any threat to someone.
Abuse:	It is for comments that are abusive in nature.
Loathe:	It describes the comments which are hateful and loathing in nature.
ID:	It includes unique Ids associated with each comment text given.
Comment text:	This column contains the comments extracted from various social media platforms.

Let's check the data now. Below I have attached the snapshot below to give an overview.

Train dataset:

```
import pandas as pd
df_train=pd.read_csv("malig_train.csv")
df_train
```

	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
...	...	...	...	...	...	...	...	...
159566	ffe987279560d7ff	":::::And for the second time of asking, when ...	0	0	0	0	0	0
159567	ffea4adeee384e90	You should be ashamed of yourself \n\nThat is ...	0	0	0	0	0	0
159568	ffee36eab5c267c9	Spitzer \n\nUmm, theres no actual article for ...	0	0	0	0	0	0
159569	fff125370e4aaaf3	And it looks like it was actually you who put ...	0	0	0	0	0	0
159570	fff46fc426af1f9a	"\nAnd ... I really don't think you understand...	0	0	0	0	0	0

159571 rows × 8 columns

Observation: Thus we see there are 159571 rows and 8 columns in the train dataset.

Test Dataset

```
#loading Test dataset
df_test=pd.read_csv("malig_test.csv")
df_test
```

	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.
...	...	...
153159	fffcd0960ee309b5	. \n i totally agree, this stuff is nothing bu...
153160	fffd7a9a6eb32c16	== Throw from out field to home plate. == \n\n...
153161	fffda9e8d6fafa9e	" \n\n == Okinotorishima categories == \n\n I ...
153162	fffe8f1340a79fc2	" \n\n == ""One of the founding nations of the...
153163	ffffce3fb183ee80	" \n :::Stop already. Your bullshit is not wel...

153164 rows × 2 columns

Thus we see there are 153164 rows and 2 columns in the test dataset

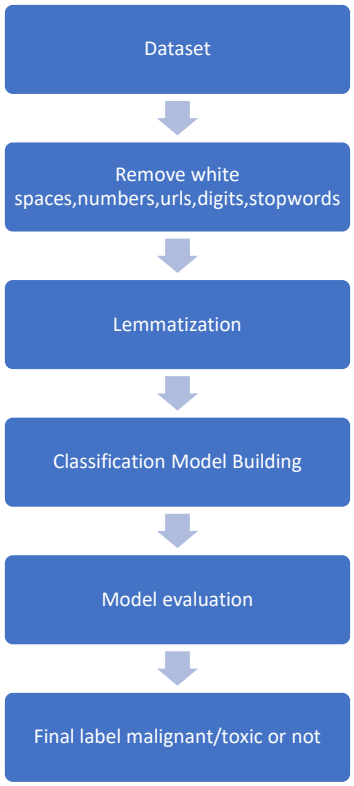
- Data Preprocessing Done

Data pre-processing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analysed by computers and machine learning**.**

Raw, real-world data in the form of text, images, video, etc., is messy. It contain errors and inconsistencies, often incomplete, and doesn't have a regular, uniform design.

Machines understands and read data as 1s and 0s. So calculating structured data, like whole numbers and percentages is easy. However, unstructured data, in the form of text and images must first be cleaned filtered and formatted before analysis.

- Steps in Data Pre-Processing



In pre-processing we started off with removing of punctuation and special character from the comments and removing column which are not important like column id. We at that point recognize that we had to also remove the stop words which are useless as they do not add any value to the dataset i.e. those comments are meaningless. Further we performed stemming and lemmatizing for the words. Lastly, we applied count vectorizer and afterward split the information into preparing and testing for the further use.

We have undergone various ways of visualizing the data to get meaningful outputs, which would help us in knowing the dataset and our final goals effectively. We tried to segregate the analysis of the dataset into various categories such as based on the word lengths, the toxic words utilized and the degree of toxicity present in them.

All this visualization helped us in gaining a wholistic picture which led us to finally sort down to two algorithms which was making our end goal in classifying the toxic comments efficiently.

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                159571 non-null  object
 1   comment_text      159571 non-null  object
 2   malignant         159571 non-null  int64
 3   highly_malignant  159571 non-null  int64
 4   rude              159571 non-null  int64
 5   threat            159571 non-null  int64
 6   abuse             159571 non-null  int64
 7   loathe            159571 non-null  int64
dtypes: int64(6), object(2)
memory usage: 9.7+ MB
```

Observation: Train:Thus we see that there are 2 columns having object datatype and 6 columns having int datatypes and no missing value present

```
df_test.dtypes
```

```
id              object
comment_text    object
dtype: object
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153164 entries, 0 to 153163
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            153164 non-null  object
 1   comment_text  153164 non-null  object
dtypes: object(2)
memory usage: 2.3+ MB
```

Thus we see there are two object datatype in test dataset.

Null values checking using heatmap for both train and test dataset.

```
#missing value
df_train.isna().sum()
```

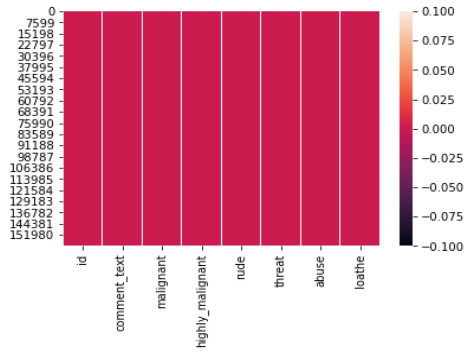
```
id                0
comment_text      0
malignant         0
highly_malignant  0
rude              0
threat            0
abuse             0
loathe            0
dtype: int64
```

```
#missing value
df_test.isna().sum()
```

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

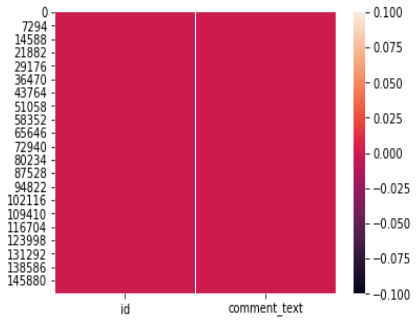
```
sns.heatmap(df_train.isnull())
```

```
<AxesSubplot:>
```




```
sns.heatmap(df_test.isnull())
```

```
<AxesSubplot:>
```



Observation: We can see that there are no null values present in the given train and test dataset.

Duplicate Rows:

```
df_train.duplicated().sum()
```

```
0
```

```
df_test.duplicated().sum()
```

```
0
```

Thus we see there are no duplicate values for train and test dataset.

```
# Lets first see how the comments look like
df_train['comment_text'][0]
```

```
"Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please don't remove the template from the talk page since I'm retired now. 89.205.38.27"
```

```
df_train['comment_text'][1]
```

```
"D'aww! He matches this background colour I'm seemingly stuck with. Thanks.  (talk) 21:51, January 11, 2016 (UTC)"
```

Observation: Thus we see that there are many words and numbers which are not important for prediction which need to be removed before model buidling.

Unique Values:

```
#printing unique features
print("Total number of unique values in each feature:")
for col in df_train.columns.values:
    print("Number of unique values of {} : {}".format(col, df_train[col].nunique()))
```

```
Total number of unique values in each feature:
Number of unique values of id : 159571
Number of unique values of comment_text : 159571
Number of unique values of malignant : 2
Number of unique values of highly_malignant : 2
Number of unique values of rude : 2
Number of unique values of threat : 2
Number of unique values of abuse : 2
Number of unique values of loathe : 2
```

Thus we see there are 6 labels having two distinct values in binary form "0" or "1".

Checking the statistical summary of the dataset

The Describe function returns the statistical summary of the dataframe or series. It Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided.

For numeric data, the result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median.

For object data (e.g. strings or timestamps), the result's index will include count, unique, top, and freq. The top is the most common value. The freq is the most common value's frequency. Timestamps also include the first and last items.

```
df_train.describe()
```

	malignant	highly_malignant	rude	threat	abuse	loathe
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
df_test.describe()
```

	id	comment_text
count	153164	153164
unique	153164	153039
top	82b910d2aff4c7fc	#NAME?
freq	1	126

Observation: Thus we see unique id is identifier so we need to remove it before model building as it has no relevance.Thus comment_text,name has been occurred 126 times .

Correlation Analysis:

Correlation analysis is a statistical method used to measure the strength of the linear relationship between two variables and compute their association.

A positive correlation indicates that the values tend to increase with one another and a negative correlation indicates that values in one set tend to decrease with an increase in the other set.

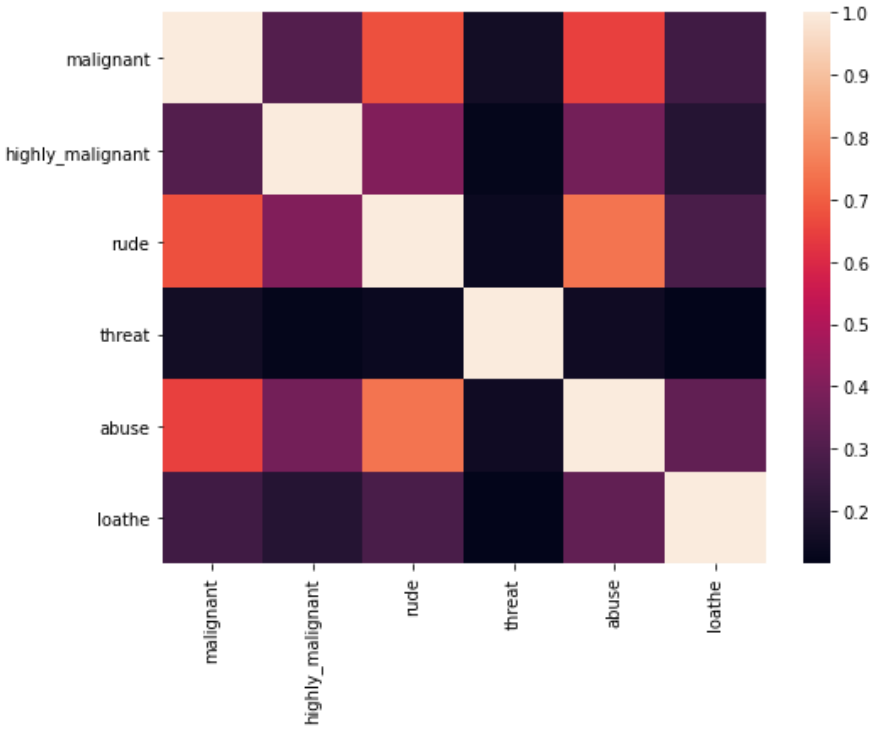
```
df_train.corr()
```

	malignant	highly_malignant	rude	threat	abuse	loathe
malignant	1.000000	0.308619	0.676515	0.157058	0.647518	0.266009
highly_malignant	0.308619	1.000000	0.403014	0.123601	0.375807	0.201600
rude	0.676515	0.403014	1.000000	0.141179	0.741272	0.286867
threat	0.157058	0.123601	0.141179	1.000000	0.150022	0.115128
abuse	0.647518	0.375807	0.741272	0.150022	1.000000	0.337736
loathe	0.266009	0.201600	0.286867	0.115128	0.337736	1.000000

Observation: Thus we see that malignant is 67% positively correlated with rude and 64% positively correlated with abuse. Also ruse is 74% positively correlated with abuse. While others are less correlated less than 40%. Interesting things to be observed is that though, highly_malignant is also a malignant comment, the correlation between them is only 0.31.

Correlation using a Heatmap

A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space**.**



Thus correlation of the variables using heatmap. Thus lighter shades are high correlated and dark shades are not correlated highly.

Feature Engineering:

Removing Column_id

```
#since the feature 'id' has no relevance w.r.t. model training therefore dropping this feature
df.drop(columns=['id'],inplace=True)
```

Conversion to Lowercase

```
#converting comment text to lowercase
df['comment_text'] = df.comment_text.str.lower()

#interpreting first 5 rows
df.head()
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	original_length
0	explanation\nwhy the edits made under my usern...	0	0	0	0	0	0	264
1	d'aww! he matches this background colour i'm s...	0	0	0	0	0	0	112
2	hey man, i'm really not trying to edit war. it...	0	0	0	0	0	0	233
3	"\nmore\ni can't make any real suggestions on ...	0	0	0	0	0	0	622
4	you, sir, are my hero. any chance you remember...	0	0	0	0	0	0	67

we see that the comment_text has been converted to lower case letters

```
#importing NLTK libraries
import nltk
from nltk.tokenize import word_tokenize, regexp_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
```

Handling unwanted characters:

```
# Removing and Replacing unwanted characters in the comment_text column

# Replacing '\n' with ' '
df.comment_text = df.comment_text.str.replace('\n',' ')
we see that the comment_text has been converted to lower case letters
# Keeping only text with letters a to z, 0 to 9 and words like can't, don't, couldn't etc
df.comment_text = df.comment_text.apply(lambda x: ' '.join(regexp_tokenize(x,"[a-z']+")))

# Removing Stop Words and Punctuations

# Getting the list of stop words of english language as set
stop_words = set(stopwords.words('english'))

# Updating the stop_words set by adding letters from a to z
for ch in range(ord('a'),ord('z')+1):
    stop_words.update(chr(ch))

# Updating stop_words further by adding some custom words
custom_words = ("d'aww","mr","hmm","umm","also","maybe","that's","he's","she's","i'll","he'll","she'll","us",
                "ok","there's","hey","heh","hi","oh","bbq","i'm","i've","nt","can't","could","ur","re","ve",
                "rofl","lol","stfu","lmk","ily","yolo","smh","lmfao","nvm","ikr","ofc","omg","ilu")
stop_words.update(custom_words)

# Checking the new list of stop words
print("New list of custom stop words are as follows:\n\n")
print(stop_words)
```

```
#Removing stop words
df.comment_text = df.comment_text.apply(lambda x: ' '.join(word for word in x.split() if word not in stop_words).strip())

#Removing punctuations
df.comment_text = df.comment_text.str.replace("[^\w\d\s]","")

df.sample(15)
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	original_length
110202	ancient china criminals guilty proven innocent...	0	0	0	0	0	0	736
157351	trying add anything wikipedia sources yet hear...	0	0	0	0	0	0	202
100425	fact make campaign stop lies machchunk frieeee...	1	0	0	0	0	0	136
7147	clearly understand conesus means every editor ...	0	0	0	0	0	0	402
113412	page provided page provided wikipedia editors ...	0	0	0	0	0	0	1312
35807	excuses keeping content makes look like circum...	0	0	0	0	0	0	204
116511	please vandalize pages edit insanegumby contin...	0	0	0	0	0	0	147
611	name yuen lou looks variation yuen lo name jac...	0	0	0	0	0	0	128
49518	see article iranica	0	0	0	0	0	0	39
86766	word alibi replaced	0	0	0	0	0	0	40
101034	november utc yeah guess including citations pr...	0	0	0	0	0	0	392
26371	think whole thing deleted pointless thelaststand	0	0	0	0	0	0	87
81012	new entity uwm created put uw system time madi...	0	0	0	0	0	0	853
38082	ravenloft stone prophet thanks adding citation...	0	0	0	0	0	0	205
17062	mar utc	0	0	0	0	0	0	24

Observation : Thus we have removed characters like '\n',converted to lowercase,digits,punctuation and stopwords. Stop words are a set of commonly used words in a language. Examples of stop words in English are "a", "the", "is", "are" and etc. Stop words are commonly used in Text Mining and Natural Language Processing (NLP) to eliminate words that are so commonly used that they carry very little useful information.

STEMMERS:

```
#Stemming words
snb_stem = SnowballStemmer('english')
df.comment_text = df.comment_text.apply(lambda x: ' '.join(snb_stem.stem(word) for word in word_tokenize(x)))
df.sample(15)
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	original_length
123191	welcom talk	0	0	0	0	0	0	21
15537	newslett sorri long delay repli messag like ne...	0	0	0	0	0	0	277
9580	actual know say pikachu lot pikachus defin num...	0	0	0	0	0	0	147
55132	boy juli walter revert mistak think disast tho...	0	0	0	0	0	0	580
19458	excel review treat section blood recent pmid i...	0	0	0	0	0	0	459
48125	mere specul	0	0	0	0	0	0	27
70655	yes must ban much must might use encyclopaedia...	0	0	0	0	0	0	172
63676	googi steampunk architectur disput greet pleas...	0	0	0	0	0	0	213
122362	dont gay hitler form hero	1	0	1	0	1	0	41
43682	aristotl work classif anim interet underdevelo...	0	0	0	0	0	0	750
37303	person support delet one content coincident re...	0	0	0	0	0	0	321
150360	remov specious content read complet sourc cite...	0	0	0	0	0	0	253
67646	put post home page time	0	0	0	0	0	0	55
69494	peopl cattl	0	0	0	0	0	0	51
50751	mov view attack fact insult anyon figur meant	0	0	0	0	0	0	449

Observation: Stemming: It is the process of reducing the word to its word stem that affixes to suffixes and prefixes or to roots of words known as a lemma. In simple words stemming is reducing a word to its base word or stem in such a way that the words of similar kind lie under a common stem. For example – The words care, cared and caring lie under the same stem 'care'. Stemming is important in natural language processing(NLP).

```
# Checking the length of comment_text after cleaning and storing it in cleaned_length variable
df["cleaned_length"] = df.comment_text.str.len()
df.head(10)
```

	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	original_length	cleaned_length
0	explan edit made usernam hardcor metallica fan...	0	0	0	0	0	0	264	135
1	match background colour seem stuck thank talk ...	0	0	0	0	0	0	112	57
2	man realli tri edit war guy constant remov rel...	0	0	0	0	0	0	233	112
3	make real suggest improv wonder section statis...	0	0	0	0	0	0	622	310
4	sir hero chanc rememb page	0	0	0	0	0	0	67	26
5	congratul well use tool well talk	0	0	0	0	0	0	65	33
6	cocksuck piss around work	1	1	1	0	1	0	44	25
7	vandal matt shirvington articl revert pleas ban	0	0	0	0	0	0	115	47
8	sorri word nonsens offens anyway intend write ...	0	0	0	0	0	0	472	235
9	align subject contrari dulithgow	0	0	0	0	0	0	70	32

Observation: Thus we see cleaned_length after removing the unwanted charters from the comment_text. We can clearly compare the original_length with cleaned_length thus see the characters removed from dataset .

```
# Checking the percentage of length cleaned
print(f"Total Original Length        : {df.original_length.sum()}")
print(f"Total Cleaned Length         : {df.cleaned_length.sum()}")
print(f"Percentage of Length Cleaned : {(df.original_length.sum()-df.cleaned_length.sum())*100/df.original_length.sum()}%")
```

```
Total Original Length        : 62893130
Total Cleaned Length         : 34297506
Percentage of Length Cleaned : 45.46700728680541%
```

- ## Data Inputs- Logic- Output Relationships

The train dataset consists of multilabel and features. The features are independent and label is dependent as the values of our independent variables changes as our label varies.

The count plots was to check the counts available in each column as a part of univariate analysis.We have used wordcloud which gives the words frequency in the labels and thus showed us which are the loud words in every label.

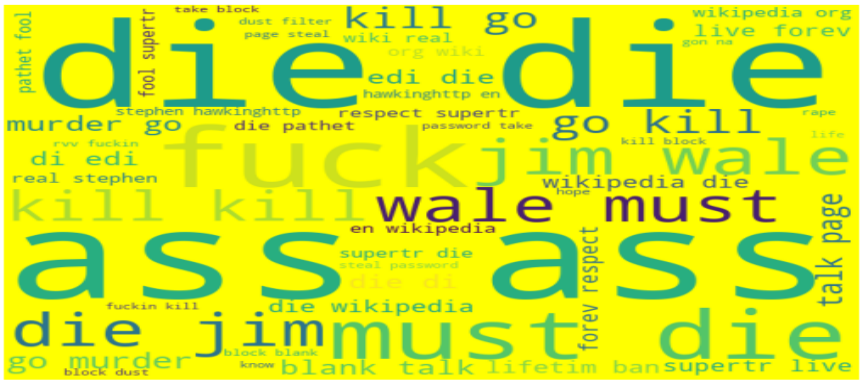
The correlation between the label and features was checked using the heat map.

Visualization of the different output labels using wordcloud:

WORD CLOUDS FOR MALIGNANT COMMENTS

WORD CLOUDS FOR RUDE COMMENTS

WORD CLOUDS FOR THREAT COMMENTS



WORD CLOUDS FOR ABUSE COMMENTS

- State the set of assumptions (if any) related to the problem under consideration

1.The max_features have been taken at 2000,if I am using values above 2000 than that there is space problem.

Hardware and Software Requirements and Tools Used

Hardware Used:

i. RAM: 8 GB
ii. CPU: Intel® Core™ i3-1005G1 CPU @1.20GHz
iii. GPU: Intel® UHD Graphics

Software Used:

i. Programming language: Python
ii. Distribution: Anaconda Navigator
iii. Browser based language shell: Jupyter Notebook

Libraries/Packages Used:

1. Pandas- a library which is used to read the data, visualisation and analysis of data.

2. NumPy- used for working with array and various mathematical techniques.

3. Seaborn- visualization tool for plotting different types of plot.

4. Matplotlib- It provides an object-oriented API for embedding plots into applications.

5. Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib

6.joblib: It is part of the SciPy ecosystem and provides utilities for pipelining Python jobs.It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

The objective of the case study is to build a model 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.
Thus this is clearly a binary multilabel classification problem.As this is a binary classification problem, so we will be loading all the libraries related to it .Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows − Before applying algorithms we will have to clean and preprocess the data. Then we will have to split the dataset into training model and testing model. Only then we can apply algorithms on it.

In this project there were 6 features which defines the type of comment like malignant, hate, abuse, threat, loathe. In this NLP based project we need to predict the multiple labels which are binary. We need to convert text into feature vectors using TF-IDF vectorizer and separate out feature and labels. Also, before building the model, it is made sure that the input data is cleaned and scaled before it was fed into the machine learning models.

- ## Testing of Identified Approaches (Algorithms)

  Since it’s a classification problem, the following algorithms have been used for model building:

  The algorithms used on training the data are as follows:

  1. Logistic Regression

  2. MultinomialNB

  3. LightGBM Classifier

  4. LinearSVC

  5. Passive Aggressive Classifier

  6.SDGClassifier

  Brief description of the selected models:

Model used:

Logistic Regression:
One of the most useful machine learning techniques in the world of statistics is logistic regression. It can be used to solve problems with binary and multi-class categorization. Because it can generate probabilities and classify new data using both continuous and discrete datasets, logistic regression is a key machine learning approach.

MultinomialNB: The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. The Multinomial Naive Bayes algorithm is a Bayesian learning approach popular in Natural Language Processing (NLP). The program guesses the tag of a text, such as an email or a newspaper story, using the Bayes theorem. It calculates each tag's likelihood for a given sample and outputs the tag with the greatest chance

LightGBM Classifier: LightGBM is a fast, distributed, high performance gradient boosting framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. LightGBM classifier helps while dealing with classification problems.

LinearSVC: The Linear Support Vector Classifier (SVC) method applies a linear kernel function to perform classification and it performs well with a large number of samples.The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

Passive Aggressive Classifier: Passive-Aggressive algorithms are generally used for large-scale learning. This is very useful in situations where there is a huge amount of data and it is computationally infeasible to train the entire dataset because of the sheer size of the data.

Passive-Aggressive algorithms are called so because:
Passive: If the prediction is correct, keep the model and do not make any changes. i.e., the data in the example is not enough to cause any changes in the model.
Aggressive: If the prediction is incorrect, make changes to the model. i.e., some change to the model may correct it.

SDGClassifier : Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

- ## Run and Evaluate selected models

The algorithms used on training the data are as follows:

1. Logistic Regression
2. MultinomialNB
3. LightGBM Classifier
4. LinearSVC
5. Passive Aggressive Classifier

6.SDGClaasifier

Building Machine Learning Models and evaluation metrics

Models:

We are going to use LogisticRegression, MultinomialNB, LightGBM Classifier, LinearSVC, Passive Aggressive Classifier and SDGClassifier for finding out the best model among those.

Evaluation metric:

classification_report, confusion_matrix, roc_curve, accuracy_score, roc_auc_score, log_loss,hamming_loss.

I created a classification function that included the evaluation metrics details for the generation of our Classification Machine Learning models.

Data Preparation for Model Training and Testing

```
]: # Converting text to vectors using TfidfVectorizer
   tfidf = TfidfVectorizer(max_features=2000)
   features = tfidf.fit_transform(df.comment_text).toarray()

   # Checking the shape of features
   features.shape
```

```
]: (159571, 2000)
```

TF_IDF Vectoriser:The tfidf_vectorizer class implements a vectorizer for calculating term frequency/inverse document frequency (TF/IDF), which can be used as a measure for modeling the importance of terms within documents. It takes in raw texts and returns an array of feature vectors

for each input text.In simple words, TFIDF is a numerical statistic that shows the importance of a word in a text document.

```
# input variables
x = features

# output variables
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)
```

Model building

```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_size=.33)
```

```
from sklearn.svm import LinearSVC
from lightgbm import LGBMClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression,PassiveAggressiveClassifier

# Model selection libraries...
from sklearn.model_selection import cross_val_score as cvs, cross_val_predict, train_test_split
from sklearn.model_selection import GridSearchCV


# Importing some metrics we can use to evaluate our model performance....
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import multilabel_confusion_matrix,hamming_loss,log_loss
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import log_loss


#splitting the data into train and test set
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.33, random_state = 42)

from sklearn.multiclass import OneVsRestClassifier
```

```
# Creating instances for different Classifiers
svc = LinearSVC()
lr = LogisticRegression(solver='lbfgs')
mnb = MultinomialNB()
lgb = LGBMClassifier()
sgd = SGDClassifier()
pac=PassiveAggressiveClassifier()


#function for printing score
def print_score(y_pred,clf):
    print('classifier:',clf.__class__.__name__)
    print("Accuracy score: {}".format(accuracy_score(y_test,y_pred)))
    print("f1_score: {}".format(f1_score(y_test,y_pred,average='micro')))
    print("Precision : ", precision_score(y_test,y_pred,average='micro'))
    print("Recall: {}".format(recall_score(y_test,y_pred,average='micro')))
    print("Hamming loss: ", hamming_loss(y_test,y_pred))
    print("Log_loss :",log_loss(y_test,y_pred))
    print("Confusion matrix:\n ", multilabel_confusion_matrix(y_test,y_pred))
    print("Classification Report:\n",classification_report(y_test,y_pred))
    print('**********************************************************\n')

#models with evaluation using OneVsRestClassifier
for classifier in [svc,lr,mnb,lgb,sgd,pac]:
    clf = OneVsRestClassifier(classifier)
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    print_score(y_pred, classifier)
```

Result:-

```
classifier: LinearSVC
Accuracy score: 0.9196148806471828
f1_score: 0.6989039983762939
Precision :  0.8606598350412397
Recall: 0.5883307705450197
Hamming loss:  0.018781214986991777
Log_loss : 1.4647268046284174
```

```
Confusion matrix:
   [[[47183   393]
   [ 1860  3223]]

  [[52064    69]
   [  419   107]]

  [[49613   215]
   [  878  1953]]

  [[52496    11]
   [  127    25]]

  [[49640   376]
   [ 1182  1461]]

  [[52137    51]
   [  353   118]]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.63      0.74      5083
           1       0.61      0.20      0.30       526
           2       0.90      0.69      0.78      2831
           3       0.69      0.16      0.27       152
           4       0.80      0.55      0.65      2643
           5       0.70      0.25      0.37       471

   micro avg       0.86      0.59      0.70     11706
   macro avg       0.76      0.42      0.52     11706
weighted avg       0.85      0.59      0.69     11706
 samples avg       0.06      0.05      0.05     11706
```

```
classifier: LogisticRegression
Accuracy score: 0.9191591180994702
f1_score: 0.6839539466860285
Precision :  0.8703801478352693
Recall: 0.5633008713480266
Hamming loss:  0.0192876178177836
Log_loss : 1.46205794402178
```

```
Confusion matrix:
  [[[47258   318]
  [ 1975  3108]]

 [[52030   103]
  [  397   129]]

 [[49647   181]
  [  985  1846]]

 [[52498     9]
  [  129    23]]

 [[49685   331]
  [ 1253  1390]]

 [[52148    40]
  [  373    98]]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.61	0.73	5083
1	0.56	0.25	0.34	526
2	0.91	0.65	0.76	2831
3	0.72	0.15	0.25	152
4	0.81	0.53	0.64	2643
5	0.71	0.21	0.32	471
micro avg	0.87	0.56	0.68	11706
macro avg	0.77	0.40	0.51	11706
weighted avg	0.86	0.56	0.68	11706
samples avg	0.06	0.05	0.05	11706

```
classifier: MultinomialNB
Accuracy score: 0.9129303632807307
f1_score: 0.6057988895743368
Precision :  0.8817959183673469
Recall: 0.46138732274047495
Hamming loss:  0.022246909360223322
Log_loss : 1.4494882320915998
```

```
Confusion matrix:
  [[[47420   156]
  [ 2576  2507]]

 [[52021   112]
  [  387   139]]

 [[49682   146]
  [ 1302  1529]]

 [[52506     1]
  [  152     0]]

 [[49746   270]
  [ 1471  1172]]

 [[52149    39]
  [  417    54]]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.49	0.65	5083
1	0.55	0.26	0.36	526
2	0.91	0.54	0.68	2831
3	0.00	0.00	0.00	152
4	0.81	0.44	0.57	2643
5	0.58	0.11	0.19	471
micro avg	0.88	0.46	0.61	11706
macro avg	0.63	0.31	0.41	11706
weighted avg	0.86	0.46	0.60	11706
samples avg	0.04	0.04	0.04	11706

```
classifier: LGBMClassifier
Accuracy score: 0.9170132361039898
f1_score: 0.697708868259925
Precision :  0.8267025509010063
Recall: 0.6035366478728857
Hamming loss:  0.01937623831317217
Log_loss : 1.284856921974861
Confusion matrix:
```

```
Confusion matrix:
  [[[47177   399]
  [ 1933  3150]]

 [[52012   121]
  [  414   112]]

 [[49534   294]
  [  797  2034]]

 [[52467    40]
  [  117    35]]

 [[49455   561]
  [ 1048  1595]]

 [[52122    66]
  [  332   139]]]
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.62      0.73      5083
           1       0.48      0.21      0.30       526
           2       0.87      0.72      0.79      2831
           3       0.47      0.23      0.31       152
           4       0.74      0.60      0.66      2643
           5       0.68      0.30      0.41       471

   micro avg       0.83      0.60      0.70     11706
   macro avg       0.69      0.45      0.53     11706
weighted avg       0.82      0.60      0.69     11706
 samples avg       0.05      0.05      0.05     11706

************************************************************


classifier: SGDClassifier
Accuracy score: 0.9153231166562221
f1_score: 0.6380010982976386
Precision :  0.8931426814268143
Recall: 0.4962412438065949
Hamming loss:  0.020863796628623154
Log_loss : 1.4676925366453593
```

```
Confusion matrix:
  [[[47388   188]
  [ 2390  2693]]

 [[52133     0]
  [  526     0]]

 [[49649   179]
  [ 1029  1802]]

 [[52507     0]
  [  152     0]]

 [[49694   322]
  [ 1365  1278]]

 [[52182     6]
  [  435    36]]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.53	0.68	5083
1	0.00	0.00	0.00	526
2	0.91	0.64	0.75	2831
3	0.00	0.00	0.00	152
4	0.80	0.48	0.60	2643
5	0.86	0.08	0.14	471
micro avg	0.89	0.50	0.64	11706
macro avg	0.58	0.29	0.36	11706
weighted avg	0.84	0.50	0.62	11706
samples avg	0.05	0.04	0.04	11706

```
  ************************************************************

classifier: PassiveAggressiveClassifier
Accuracy score: 0.9033973299910747
f1_score: 0.6866379689009927
Precision :  0.7066907775768535
Recall: 0.6676917819921407
Hamming loss:  0.022579236217930457
Log_loss : 1.2295544430788288
```

```
Confusion matrix:
  [[[46830   746]
  [ 1695  3388]]

 [[51194   939]
  [  159   367]]

 [[49267   561]
  [  681  2150]]

 [[52480    27]
  [  116    36]]

 [[49071   945]
  [  842  1801]]

 [[52162    26]
  [  397    74]]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.67	0.74	5083
1	0.28	0.70	0.40	526
2	0.79	0.76	0.78	2831
3	0.57	0.24	0.33	152
4	0.66	0.68	0.67	2643
5	0.74	0.16	0.26	471
micro avg	0.71	0.67	0.69	11706
macro avg	0.64	0.53	0.53	11706
weighted avg	0.75	0.67	0.69	11706
samples avg	0.06	0.06	0.05	11706

Observation: From the above model comparison it is clear that Linear Support Vector Classifier performs better with Accuracy Score: 91.96% and Hamming Loss: 0.018% and F1_score is 70% than the other classification models,we are considering specially F1 score as the basis of evaluation of best model,as f1_score can goes well with imbalance dataset.Therefore We will use Linear Support Vector Classifier for further Hyperparameter tuning process.

HYPERPARAMETER TUNNING

```
parameters = {
        'estimator__penalty': ['l1'],
        'estimator__loss': ['hinge','squared_hinge'],
        'estimator__multi_class': ['ovr','crammer_singer'],
        'estimator__intercept_scaling': [2,4,5],
         }
#train the model with given parameters using GridSearchCV
svc = OneVsRestClassifier(LinearSVC())
GCV =  GridSearchCV(svc,parameters,cv = 2, verbose =0,n_jobs=-1)
GCV.fit(x_train,y_train)

#printing the best parameters found by GridSearchCV
GCV.best_params_
```

```
{'estimator__intercept_scaling': 2,
 'estimator__loss': 'hinge',
 'estimator__multi_class': 'crammer_singer',
 'estimator__penalty': 'l1'}
```

Observation: Thus on 4 parameters have been used under the variable parameters that is used in GridSearchCV for finding the best result. we see the best parameters which will help in improving the performance of the model using GridSearchCV. Then we use this best parameters to the GridSearchCV. function.

FINAL MODEL

```
flmodel = OneVsRestClassifier(LinearSVC(loss='hinge',multi_class='crammer_singer', penalty ='l1',intercept_scaling=2))
flmodel.fit(x_train,y_train)
y_pred = flmodel.predict(x_test)

print("Accuracy score: {}".format(accuracy_score(y_test,y_pred)))
roc_auc=roc_auc_score(y_test,y_pred)
print("roc_auc_score",roc_auc*100)
print("f1_score: {}".format(f1_score(y_test,y_pred,average='micro')))
print("Precision : ", precision_score(y_test,y_pred,average='micro'))
print("Recall: {}".format(recall_score(y_test,y_pred,average='micro')))
print("Hamming loss: ", hamming_loss(y_test,y_pred))
print("Log_loss :",log_loss(y_test,y_pred))
print("\nConfusion matrix: \n", multilabel_confusion_matrix(y_test,y_pred))

flmodel.predict(x)
```

```
Accuracy score: 0.9177158700317135
roc_auc_score 67.11714849428851
f1_score: 0.6896869716686307
Precision :  0.8615128631767567
Recall: 0.5750042713138561
Hamming loss:  0.01917051216316299
Log_loss : 1.535319085721115
```

```
Confusion matrix:
 [[[47185   391]
  [ 1881  3202]]

 [[52132     1]
  [  519     7]]

 [[49587   241]
  [  873  1958]]

 [[52507     0]
  [  149     3]]

 [[49588   428]
  [ 1157  1486]]

 [[52167    21]
  [  396    75]]]
```

Observation:

Thus our final models shows the accuracy score of 91.77%,f1_score of 69%,roc_auc_score 67.12%, Precision:86.15%,Recall:57.50%,Hamming loss:0.019,Log_loss : 1.53.

Hamming loss: Hamming loss is the fraction of wrong labels to the total number of labels. In multi-class classification, hamming loss is calculated as the hamming distance between y_true and y_pred.It is always between 0 and 1, lower being better.

Log_loss:Log Loss is the most important classification metric based on probabilities. For any given problem, a lower log loss value means better predictions.Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value.

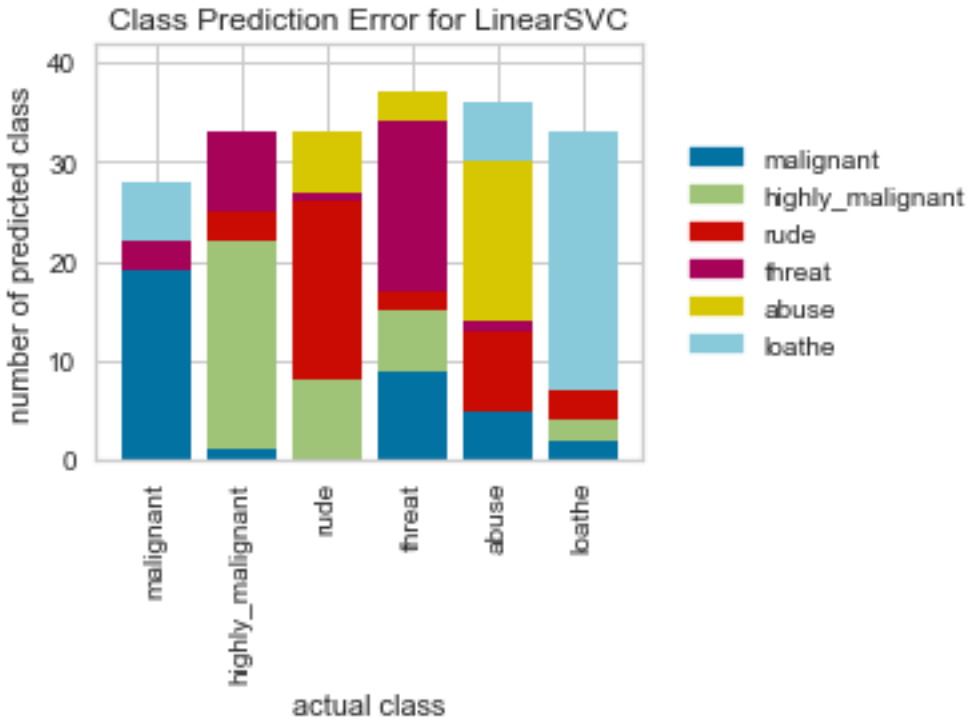
Class Prediction Error:

The Yellowbrick ClassPredictionError plot is a twist on other and sometimes more familiar classification model diagnostic tools like the Confusion Matrix and Classification Report. Like the Classification Report, this plot shows the support (number of training samples) for each class in the fitted classification model as a stacked bar chart. Each bar is

segmented to show the proportion of predictions (including false negatives and false positives, like a Confusion Matrix) for each class. We can use a ClassPredictionError to visualize which classes the classifier is having a particularly difficult time with, and more importantly, what incorrect answers it is giving on a per-class basis. This can often enable us to better understand strengths and weaknesses of different models and particular challenges unique to the dataset.

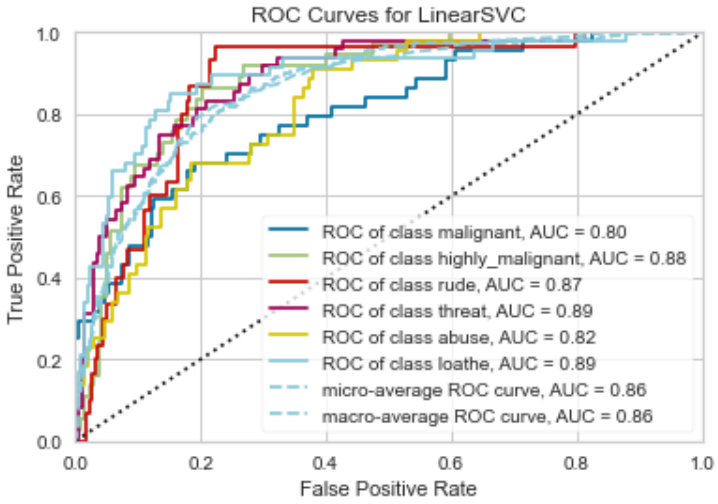
Class Prediction Error

5]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from yellowbrick.classifier import ClassPredictionError


# Create classification dataset
x, y = make_classification(
    n_samples=1000, n_classes=6, n_informative=3, n_clusters_per_class=1,
    random_state=36,
)

classes = ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']

# Perform 80/20 training/test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
                                                    random_state=42)
# Instantiate the classification model and visualizer
visualizer = ClassPredictionError(
    LinearSVC(random_state=42), classes=classes
)

# Fit the training data to the visualizer
visualizer.fit(x_train, y_train)

# Evaluate the model on the test data
visualizer.score(x_test, y_test)

# Draw visualization
visualizer.show()
```



Observation: The class prediction error chart provides a way to quickly understand how good your classifier is at predicting the right classes.
Thus in the above example, while the LinearSVC appears to be fairly good at correctly predicting malignant based on the features of the toxic comments, it often incorrectly labels rude as hihly_malignant and mistakes hihly malignant for loathe.Using these new insights, one could easily detect the features causing issues and quickly address the problem

AUC_ROC CURVE FOR Final model



Observation: ROC_AUC is generally used for binary classification, however, Yellowbrick's ROC_AUC does allow for multi-class classification. It did a better job displaying ROC_AUC curve. The graph displays the ROC for each class as well as the micro and macro averages. The micro averages are computed from the sum of all true positives and false positives across all classes and the macro averages are the averages of curves across all classes.

ROC curves are typically used in binary classification, and in fact the Scikit-Learn roc_curve metric is only able to perform metrics for binary classifiers. As a result, it is necessary to binarize the output or to use one-vs-rest or one-vs-all strategies of classification. The visualizer does its best to handle multiple situations, but exceptions can arise from unexpected models or outputs.

Thus we can see for malignant, auc_roc score is 80%,for highly_malignant,its 88%,for class rude ,its 87%,for class threat-it is showing 89%,for class abuse, it is 82% and for class loathe is 89%.Thus it is well able to distinguish toxic comments for the different classes.

Saving the Model

```
import joblib
joblib.dump(flmodel,"Malignant_comment_classifier.pkl")
#Loading the model
model = joblib.load('Malignant_comment_classifier.pkl')
```

I am using joblib to save the final classification model.

- ## Key Metrics for success in solving problem under consideration

The key metrics used here were Accuracy Score, Precision, Recall, F1 score, Roc Auc Score ,Hamming Loss,Log Loss and Confusion Matrix. We tried to find out the best parameters and also to increase our scores by using Hyperparameter Tuning and used GridSearchCV method

1.Accuracy Score Model accuracy is a machine learning model performance metric that is defined as the ratio of true positives and true

negatives to all positive and negative observations. In other words, accuracy tells us how often we can expect our machine learning model will correctly predict an outcome out of the total number of times it made predictions.

Mathematically, it represents the ratio of the sum of true positive and true negatives out of all the predictions.

Accuracy Score = (TP + TN)/ (TP + FN + TN + FP)

2.Precision: The model precision score measures the proportion of positively predicted labels that are actually correct. Precision is also known as the positive predictive value. Precision is used in conjunction with the recall to trade-off false positives and false negatives. Precision is affected by the class distribution. If there are more samples in the minority class, then precision will be lower. Precision can be thought of as a measure of exactness or quality

The precision score is a useful measure of the success of prediction when the classes are very imbalanced. **Mathematically, it represents the ratio of true positive to the sum of true positive and false positive.**

Precision Score = TP / (FP + TP)

3.Recall: Model recall score represents the model's ability to correctly predict the positives out of actual positives. This is unlike precision which measures how many predictions made by models are actually positive out of all positive predictions made. For example: If your machine learning model is trying to identify positive reviews, the recall score would be what percent of those positive reviews did your machine learning model correctly predict as a positive. In other words, it measures how good our machine learning model is at identifying all actual positives out of all positives that exist within a dataset. The higher the recall score, the better the machine learning model is at identifying both positive and negative examples. Recall is also known as sensitivity or the true positive rate. A high recall score indicates that the model is good at identifying positive examples.

Mathematically, it represents the ratio of true positive to the sum of true positive and false negative.

Recall Score = TP / (FN + TP)

4.F1: Model F1 score represents the model score as a function of precision and recall score. F-score is a machine learning model performance metric that gives equal weight to both the Precision and Recall for measuring its performance in terms of accuracy, making it an alternative to Accuracy metrics (it doesn't require us to know the total number of observations). It's often used as a single value that provides high-level information about the model's output quality. This is a useful measure of the model in the scenarios where one tries to optimize either of precision or recall score and as a result, the model performance suffers. The F1 score becomes especially valuable when working on classification models in which your data set is imbalanced.

Mathematically, it can be represented as a harmonic mean of precision and recall score.

F1 Score = 2* Precision Score * Recall Score/ (Precision Score + Recall Score/)

5. Roc Auc Score: The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values. The Area Under Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

6.Confusion Matrix is one of the evaluation metrics for machine learning classification problems, where a trained model is being evaluated for accuracy and other performance measures. And this matrix is called the confusion matrix since it results in an output that shows how the system is confused between the two classes.

7.Hamming loss: Hamming loss is the fraction of wrong labels to the total number of labels. In multi-class classification, hamming loss is calculated as the hamming distance between y_true and y_pred.It is always between 0 and 1, lower being better.

8.Log_loss:Log Loss is the most important classification metric based on probabilities. For any given problem, a lower log loss value means better

predictions.Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value.

9.Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning. We must select from a specific list of hyperparameters for a given model as it varies from model to model.

Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set.

Using Scikit-Learn's GridSearchCV method: GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

Model Analysis using Hyperparameter:

Thus our final models shows the accuracy score of 91.77%, f1_scoreof69% , roc_auc_score of 67.12%, haing precision score of :86.15%,having recall score of :57.50%,Hamming loss:0.019,Log_loss : 1.53.Thus LinearSVC is the best model chosen for further deployment process.

Prediction for test dataset using final model

```
df_test.drop(columns=['id'],inplace=True)
```

```
dft = df_test.copy()
dft['original_length'] = df_test.comment_text.str.len()

dft.head()
```

First we need to data clean the test data as we did in the train dataset.

```
# Replacing '\n' with ' '
dft.comment_text = dft.comment_text.str.replace('\n',' ')

# Keeping only text with letters a to z, 0 to 9 and words like can't, don't, couldn't etc
dft.comment_text = dft.comment_text.apply(lambda x: ' '.join(regexp_tokenize(x,"[a-z']+")))

# Removing Stop Words and Punctuations

# Getting the list of stop words of english language as set
stop_words = set(stopwords.words('english'))

# Updating the stop_words set by adding letters from a to z
for ch in range(ord('a'),ord('z')+1):
    stop_words.update(chr(ch))

# Updating stop_words further by adding some custom words
custom_words = ("d'aww","mr","hmm","umm","also","maybe","that's","he's","she's","i'll","he'll","she'll","us",
                "ok","there's","hey","heh","hi","oh","bbq","i'm","i've","nt","can't","could","ur","re","ve",
                "rofl","lol","stfu","lmk","ily","yolo","smh","lmfao","nvm","ikr","ofc","omg","ilu")
stop_words.update(custom_words)

# Checking the new list of stop words
print("New list of custom stop words are as follows:\n\n")
print(stop_words)
```

New list of custom stop words are as follows:

```
#Removing stop words
dft.comment_text = dft.comment_text.apply(lambda x: ' '.join(word for word in x.split() if word not in stop_words).strip())

#Removing punctuations
dft.comment_text = dft.comment_text.str.replace("[^\w\d\s]","")

dft.sample(15)
```

	comment_text	original_length
109894	hanks support ve made administrator ll best let	125
99984	erbia nikada nije imala adranskjo ore nikada n...	70
15791	ell good faith assumed certainly want accuse a...	904
10025	im ear hank appreciate reply im ear page	78
134991	age uestion ello trying edit yron allimores pa...	404
123326	give poor metal colour	60
60239	seem able copy paste contents block infobox in...	110
131278	think fact uses word affordable questionable f...	353
82590	white ey yo remember shit azy back otherfucker...	3638
130885	top whining correct environment capable evil	74
42690	reenhouse effect dependent location omeone rem...	299
42533	ad atties hettoville	31
82354	ock ase started sock puppet case ee ikipedia u...	104
19584	utomation eriously wanted conceal automation t...	221
118589	assassinate man	20

```
#Stemming words
snb_stem = SnowballStemmer('english')
dft.comment_text = dft.comment_text.apply(lambda x: ' '.join(snb_stem.stem(word) for word in word_tokenize(x)))
dft.sample(15)
```

	comment_text	original_length
113600	interpret comment oni way oni sometim complet ...	475
85502	utrit hould nutrit therapi includ qualiti arti...	223
135664	hat argument name call ts better far content s...	116
45933	mod gay	20
101229	hom sexual ped phile http www vice com read ev...	192
69471	nclude inform statement substanti document sta...	129
101374	efinit mea would sign four tild	53
51653	read child nexplain magazin load great thing l...	376
67015	click back page evil thing ucaspet	55
82153	peni sourc ant see	34
41701	ebruari leas stop disrupt edit continu vandal ...	259
80228	tupid ser xcuse msarmad buse ou dit mani artic...	382
150277	bye fuck	23

```
# Checking the length of comment_text after cleaning and storing it in cleaned_length variable
dft["cleaned_length"] = dft.comment_text.str.len()
dft.head(10)
```

	comment_text	original_length	cleaned_length
0	bitch ule succes ever what hate sad mofucka bi...	367	215
1	rom titl fine	50	13
2	ourc awe shton apland	54	21
3	look back sourc inform updat correct form gues...	205	91
4	anonym edit articl	41	18
5	hank understand think high would revert withou...	96	55
6	leas add nonsens ikipedia uch edit consid vand...	176	109
7	ear god site horribl	32	20
8	nli fool believ number correct number lie onde...	556	256
9	oubl edirect hen fix doubl redirect blank oute...	224	120

```
# Converting text to vectors using TfidfVectorizer
tfidf = TfidfVectorizer(max_features=2000)
features = tfidf.fit_transform(dft.comment_text).toarray()

# Checking the shape of features
features.shape
```

```
(153164, 2000)
```

```
# Load saved or serialized model and predict
model_loaded = joblib.load('Malignant_comment_classifier.pkl')
```

```
# Make predictions and view the results
predict_test = model_loaded.predict(features)
```

```
predict_test
```

```
array([[1, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       ...,
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0]])
```

```
# Saving predicted values into a CSV file
pd.DataFrame(predict_test).to_csv('Predicted_test_output.csv')
```

```
df1 = pd.read_csv('Predicted_test_output.csv')
df1
```

	Unnamed: 0	0	1	2	3	4	5
0	0	1	0	1	0	0	0
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0
3	3	0	0	0	0	0	0
4	4	1	0	1	0	0	0
...	...	...	...	...	...	...	...
153159	153159	0	0	0	0	0	0
153160	153160	1	0	1	0	0	0
153161	153161	0	0	0	0	0	0
153162	153162	0	0	0	0	0	0
153163	153163	1	0	0	0	0	0

153164 rows × 7 columns

```
df1.drop("Unnamed: 0", axis=1, inplace=True)
df1.rename({'0':'malignant', '1':'highly_malignant', '2':'rude', '3':'threat', '4':'abuse', '5':'loathe'},
           axis='columns', inplace=True)
```

```
df2=dft.copy()
dffinal = pd.concat([df2, df1], axis=1)
```

```
df2=dft.copy()
dffinal = pd.concat([df2, df1], axis=1)
dffinal
```

	comment_text	original_length	cleaned_length	malignant	highly_malignant	rude	threat	abuse	loathe
0	bitch ule succes ever what hate sad mofucka bi...	367	215	1	0	1	0	0	0
1	rom titl fine	50	13	0	0	0	0	0	0
2	ourc awe shton apland	54	21	0	0	0	0	0	0
3	look back sourc inform updat correct form gues...	205	91	0	0	0	0	0	0
4	anonym edit articl	41	18	1	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
153159	total agre stuff noth long crap	60	31	0	0	0	0	0	0
153160	hrow field home plate oe get faster throw cut ...	198	107	1	0	1	0	0	0
153161	kinotorishima categori see chang agre correct ...	423	205	0	0	0	0	0	0
153162	ne found nation ermani aw eturn quit similar s...	502	277	0	0	0	0	0	0
153163	top alreadi bullshit welcom m fool think kind ...	141	69	1	0	0	0	0	0

153164 rows × 9 columns

```
dffinal.to_csv('Malignant_predictions.csv', index=False)
```

- **Visualizations**

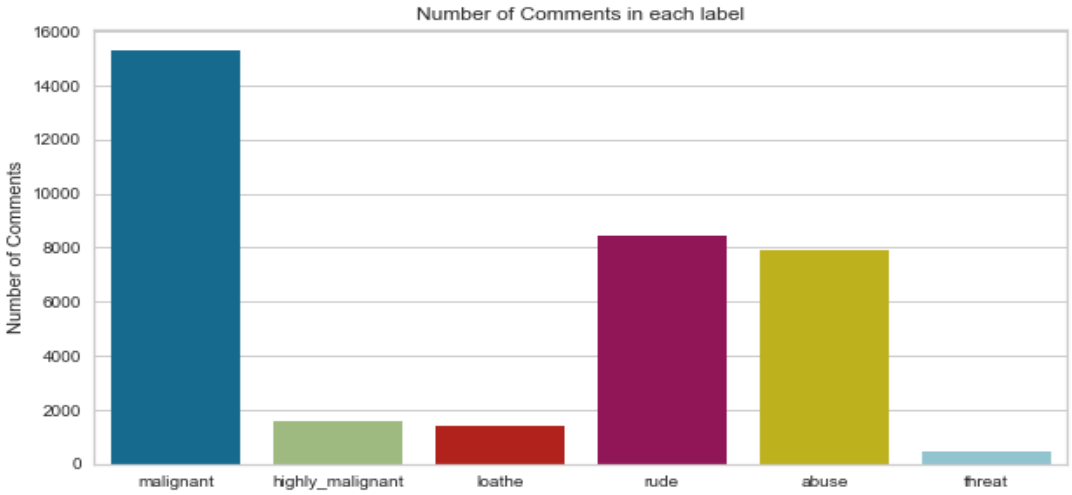
Now, we will see the different plots done with this train dataset in order to know the insight of the data present. Below are the codes given for the plots and the output obtained:

Importing required libraries :

```
#importing all the necessary libraries
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline


from scipy import interp
import scikitplot as skplt
from itertools import cycle
import matplotlib.ticker as plticker

import nltk
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import word_tokenize, regexp_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from scipy.sparse import csr_matrix

import timeit, sys
from sklearn import metrics
import tqdm.notebook as tqdm
from sklearn.svm import LinearSVC
```

Distribution of labels:

Since this is a multi label classification task, each comment can have more than one label. Here, we will see number of comments belonging to a label. Also, we will see if comments are not classified to any labels.



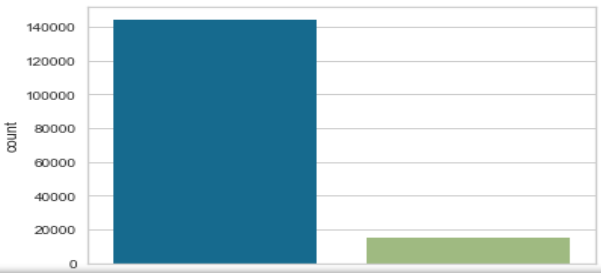
Observation:

As we see, there are higher number of comments belonging to malignant label followed by rude and abuse and comments in highly_malignant, loathe and threat are low. So, here we have the problem of imbalanced data and therefore we will have to be careful in selecting the correct evaluation metric later. We can further check to see number of labels that a comment have.

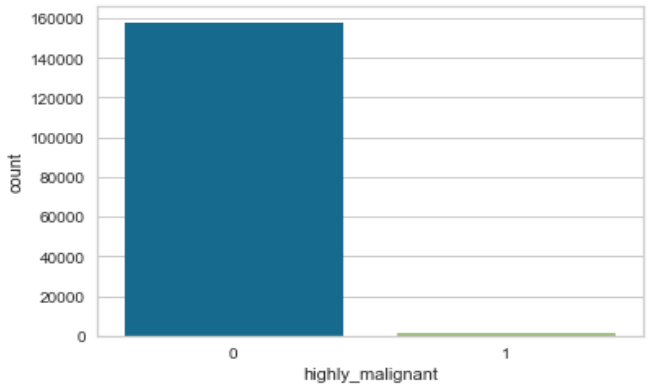
Countplot:

```
column_name=['malignant','highly_malignant','loathe','rude','abuse','threat']
for i in column_name:
    print(i)
    print("*****************")
    print(df_train[i].value_counts())
    sns.countplot(df_train[i])
    plt.show()
```

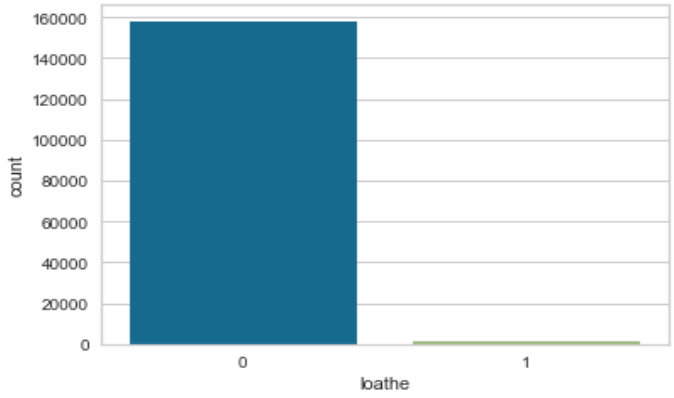
```
malignant
*****************
0    144277
1     15294
Name: malignant, dtype: int64
```



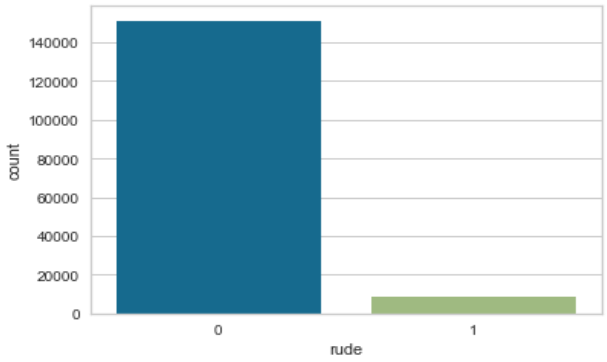
highly_malignant
****
0   157976
1   1595
Name: highly_malignant, dtype: int64



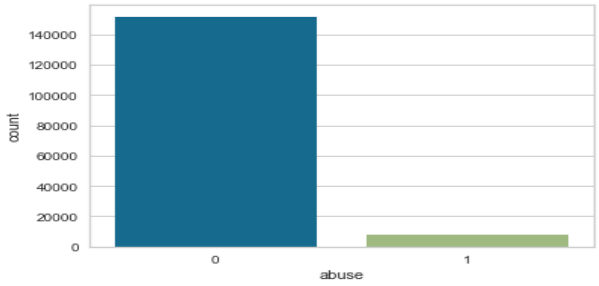
loathe
****
0   158166
1   1405
Name: loathe, dtype: int64



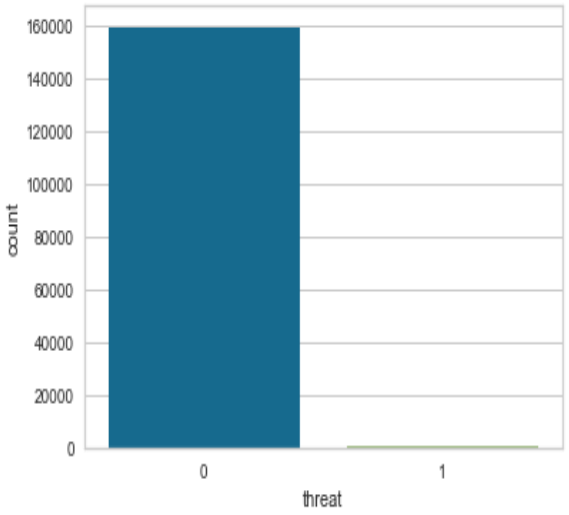
rude
****
0   151122
1   8449
Name: rude, dtype: int64




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



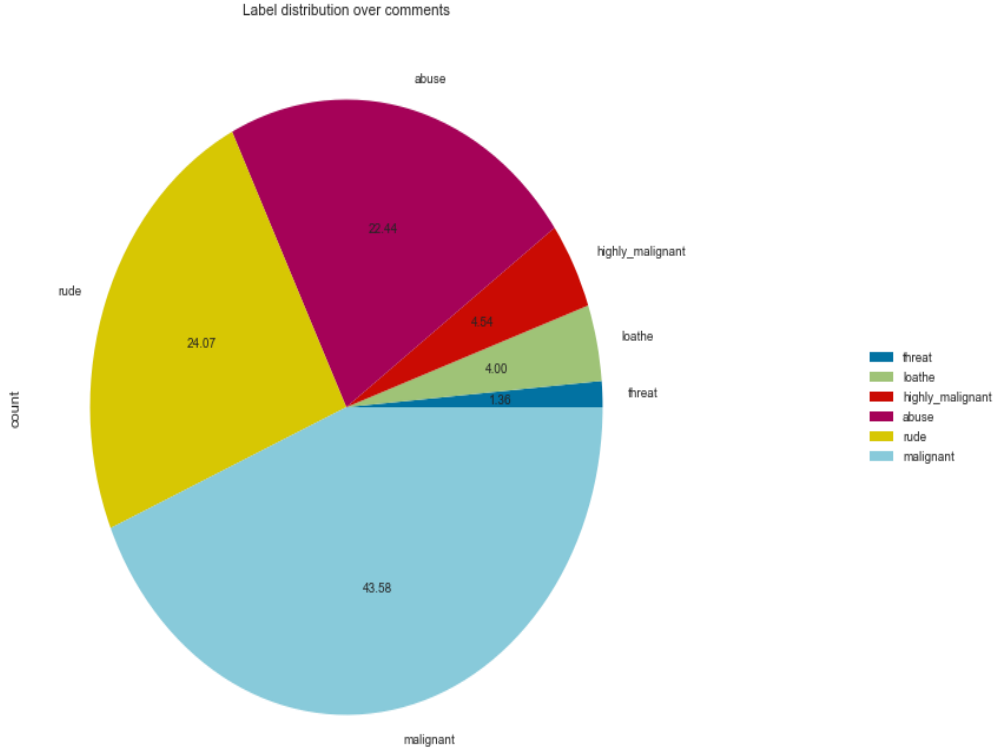

```
threat
******************
0    159093
1       478
Name: threat, dtype: int64
```



Observation: Thus we see that malignant comment has occurred 15294 times, highly_malignant comment has occurred around 1595 times, loathe comment has occurred 1405 times, rude comment has occurred 8449 times,abuse comment has occurred 7877 times and threat comment has occurred 478 times.

Pie_Plot:

```
# Visualizing the label distribution of comments using pie chart

comments_labels = ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']
df_distribution = df[comments_labels].sum()\
                        .to_frame()\
                        .rename(columns={0: 'count'})\
                        .sort_values('count')

df_distribution.plot.pie(y = 'count', title = 'label distribution over comments', autopct='%.2f', figsize = (15, 10))\
                        .legend(loc='center left', bbox_to_anchor=(1.3, 0.5))
```



Observation: Thus we see that malignant comment occupies around 44% of the total toxic comment followed by rude comment of about 24.07% followed by abuse of 22.44 % and the least is threat of about 1.36%.

Visualization of most frequent words used in the tweets:

Here we will use word cloud to visualize most frequent words used in the tweets.

Malignant Comments:

Observation: From wordcloud of malignant comments, it is clear that it mostly consists of words like fuck, nigger, moron, hate, suck etc.

Highly_malignant:

Observation: From wordcloud of highly_malignant comments, it is clear that it mostly consists of words like ass, fuck, bitch, shit, die, suck, faggot ect.

Loathe:

Observation: From wordcloud of loathe comments, it is clear that it mostly consists of words like nigga, stupid, nigger, die, gay cunt etc.

Rude:

Observation: From wordcloud of rude comments, it is clear that it mostly consists of words like nigger, ass, fuck, suck, bullshit, bitch etc.

Threat:

Observation: From wordcloud of threat comments, it is clear that it mostly consists of words like die, must die, kill, murder etc.

Abuse:

Observation: From wordcloud of abuse comments, it is clear that it mostly consists of words like moron, nigger, fat, jew, bitch etc.

CONCLUSIONS

Key Findings and Conclusions of the Study

The objective of the case was 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. We are given huge number of comments having multiple labels and also, they are in binary form.

So first I started with loading the dataset and carry out data analysis and then did the EDA process by removing all the unwanted words,stopwords etc from the toxic comments with visualization patterns using pie-plot, count plot, bar plot and visualize through wordcloud and learnt about different relationship between the features and target variable.

Then I did the model training first using TF_IDFvectoriser, I converted comments into vectors first , building the model and finding out the best model out of several classification models on the basis of different evaluation metrices scores like accuracy score, cross validation,f1,confusion matrix,auc_roc plot ,hamming loss, log loss.

We find that LinearSVC Classifier model was the best fitted model as it was having high accuracy,high F1_score and low hamming loss among other models.Then I did hyperparameter tunning through GridSearchCv on best model taking the parameters to improve the score.Also the ROC_AUC score was 90% which is very good.Thus, concluded that LinearSVC was performing well among other models.

Thus, finally concluding saying that LinearSVC was having the highest precision accuracy for prediction of toxic comments with machine learning data. Hence by implementation of this model ,

I saved the best model using joblib method and loaded the model for prediction test. Thus, this model can be used in further deployment process.

Learning Outcomes of the Study in respect of Data Science

1.Among 159,000 comments in the main database, there were about 8% of the cases that were labelled as toxic, while the remaining 92% were labelled as nontoxic

2. Harmful or toxic comments in the social media space have many negative impacts to society. The ability to readily and accurately identify comments as toxic could provide many benefits while mitigating the harm.

3.The case study pertains to cases of cyberbullying and trolls on various social media platforms having toxic and abusive comments. With the help of dataset provide to us, we came to know the which are the loud words affecting the social media space. We observe this using different visualization technique like wordcloud, bar plot, pie plot etc.

4.We knew about different Natural Language Processing techniques to be applied for text processing for data cleaning like conversion to lowercase, removal of stopwords,removal of unwated words like question mark,punctuation etc,stemming and finally converting the strings into vectors using TFIDF vectorizer for building of model.

5.Therefore in this project, we present various algorithms while predicting toxic comments using classification models with good accuracy. We tested various models such as logistic regression, LinearSVC,SDG Classifier,LightGBMClassifier,MultinomialNB,PaasiveAgrressiveClassifier and selected the best fit among models and also done hyperparameter tuning.Also we applied different evaluation metrics like hamming loss and log loss apart from accuracy.

This project directs us that it can be the best application for predicting the toxic and malignant comments using NLP and machine learning model.

Therfore we Therefore in this

Limitations of this work and Scope for Future Work

1.The dataset was imbalance and having bad comments texts.

2.Time consumption was more for model building and there was issue of space also, so I limited max_feature to 2000 instead of taking more values because there was memory issue and google_colab too,it took time and many times crashed or hanged.

2. Deep Learning architecture like LSTM, Transformers, BERT can be used to solve the same problem.

3. It can also be experimented with clustering — an unsupervised learning approach.skmultilearn.cluster module provides various clustering algorithms to solve multi-label classification.

4. We suggest a plan to improve the NLP classifiers: first by using other algorithms which such as Support Vector Clustering (SVC) and Convolutional Neural Networks (CNN).We can use more classification models to study the same.

5. We can still improve error, model accuracy with some feature engineering and by doing some extensive hyperparameter tuning on it.