MALIGNANT COMMENTS

Submitted by:

Balpreet Kaur

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped you and guided you in completion of the project:

Various online sources were used in order to get completed this project such as Analytica Vidhya, Medium, Towards Data Science and Data Trained tutorial and notes and sample of projects done in Github such as:

- https://medium.com/analytics-vidhya/jigsaw-unintended-bias-toxic-comment-classification-fa68fe3a27c8

- https://github.com/NikhilGohil/Social-Media-and-Data-Mining-Project/blob/master/SMDM%20Project%20Report.pdf-

- https://towardsdatascience.com/classifying-toxicity-in-online-comment-forums-end-to-end-project-57720af39d0b

- https://medium.datadriveninvestor.com/toxic-comment-classification-a-kaggle-case-study-a929b37150b

https://www.researchgate.net/publication/348635009_Detecting_Toxic_Remarks_in_Online_Conversations

https://www.researchgate.net/publication/349929587_Machine_learning_methods_for_toxic_comment_classification_a_systematic_review

https://www.researchgate.net/publication/334123568_Toxic_Comment_Classification

https://www.rsisinternational.org/journals/ijrias/DigitalLibrary/Vol.4&Issue11/142-147.pdf

https://github.com/tianqwang/Toxic-Comment-Classification-Challenge

https://github.com/topics/toxic-comment-classification

https://github.com/IBM/MAX-Toxic-Comment-Classifier

https://github.com/cjvegi/Toxic-Comment-Challenge

https://github.com/ProspectivePulse/Toxic-Comment-Classification-Challenge--Kaggle-Competition-

https://github.com/e-vdb/toxic-comment-classification

https://www.researchgate.net/publication/338798595_Toxic_Comment_Detection_in_Online_Discussions

https://www.researchgate.net/publication/324169011_Toxic_Comment_Tools_A_Case_Study

https://www.researchgate.net/publication/353519647_Toxic_Comment_Classification

https://www.researchgate.net/publication/326250036_Convolutional_Neural_Networks_for_Toxic_Comment_Classification

https://www.researchgate.net/publication/352180987_A_Novel_Preprocessing_Technique_for_Toxic_Comment_Classification

Here DataTrained helped me a lot in the guidance of the project.

And Data was obtained from FlipRobo Company.

Research Paper used were:

- Thesis Detecting Toxic Remarks in Online Conversations from Pushpit Gautam
- Article Machine learning methods for toxic comment classification: a systematic review from Darko Androcec
- Toxic Comment Classification from Pallam Ravi
- Toxic Comment Detection in Online Discussions from Julian Risch
- Toxic Comment Tools: A Case Study from Hetal P Patel
- Toxic Comment Classification from Sonika Prakash
- Convolutional Neural Networks for Toxic Comment Classification from Spiros V Georgakopoulos
- A Novel Preprocessing Technique for Toxic Comment Classification from : Muhammad Husnain, Adnan Khalid and Numan Shafi

INTRODUCTION

- Business Problem Framing

  The business problem is that hate and conflict throughout the online platforms makes online environments uninviting for users. 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 from spreading hatred and cyberbullying on online platforms.

- Conceptual Background of the Domain Problem

  I think having knowledge of online platforms and customers behaviour at least the way of explaining their opinions. And generally speaking, every data scientist has to have a minimum domain expertise that implies knowledge and understanding of the essential aspects of the specific field of online comments and way of sharing their opinions. In other words, a data scientist must know the stuff he/she is involved with. It is essential in order to make and have a valuable results and conclusions.

  In addition, it is essentially required for data mining and pattern discovery. The data scientist needs to know how the essentials of the specific field works so that the discovery, the patterns and evaluation process is guided by an intuitive knowledge of what actually matters both in terms of inputs and outputs as well as of what makes more sense or not.

- Review of Literature

  This is a comprehensive summary of the research done on the topic. The review should enumerate, describe, summarize, evaluate and clarify the research done.

  Hate speech is commonly defined as any communication that disparages a person or a group on the basis of some characteristics such as race, colour, ethnicity, gender, sexual orientation,

nationality, religion. Basically, you need to understand the foundation and basics of the comments and its structure. How people create and share their comments in the online platforms. For that I did some research on the following topics:

- Understand different ways of commenting on several different websites.
- How people post a hate comment
- The type of user: if anonymous or not. Most probably it's usually through anonymous user id.
- The way a user shows his/her experience and do an emotional response.
- Research a bit about the terms and conditions of several websites in order to get some idea of how comments should be shared in each websites.
- There is no universally accepted definition of hate speech either mainly because of the vague and subjective determinations or whether speech is "offensive" or conveys "hate".

- Motivation for the Problem Undertaken

  Describe your objective behind to make this project, this domain and what is the motivation behind.

  My interest started when the online platform was used and enabled to write, without any obligation to reveal oneself directly, means that this new medium of virtual communication allows people to feel greater freedom in the way they express themselves. My interest started as the Social media have become a ground for heated discussions which may frequently result in the use of insulting and offensive language since the starting the freedom of express in the internet and especially an option to state and opine as anomously. As a result, it would be good to understand and know how to recognize hate speech as a serious problem, and how it would led to develop business solutions.

Analytical Problem Framing

- Data Sources and their formats

  What are the data sources, their origins, their formats and other details that you find necessary? They can be described here. Provide a proper data description. You can also add a snapshot of the data.

```
[3]: #Read the train csv file into dataframe train
     train = pd.read_csv("train.csv")

[4]: #printing the shape of train
     print(train.shape)

     (159571, 8)

[5]: #printing a few lines of train dataset:
     train.head()
```

[5]:

	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

```
[6]: #Read the train csv file into dataframe train
     train = pd.read_csv("train.csv")
     print(train.shape)
     train.head()

     (159571, 8)
```

[6]:

	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

In the training dataset, we have a comment_text, a id for each comment and each comment is defined as malignant, highly_malignant, rude, threat, abuse and/or loathe.

```
[7]: #Read the test csv file into dataframe test and prnting shape and few data instances sample
test = pd.read_csv("test.csv")
print(test.shape)
test.head()
```

```
(153164, 2)
```

[7]:

	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.

In our testing set, we have comment id and the comment itself.

Notice that the training data contains 159,571 observations with 8 columns and the test data contains 153,164 observations with 2 columns.

```
[8]: #List the data type of each feature for training and test dataset:
print(train.dtypes)
print(test.dtypes)
```

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

As we see, the comment id and comment itself are the 2 features whose data type is object.

The remaining ones, the target columns, are the numerical ones.

```
[5]: 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
```

Here we can see the training dataset have 159571 cases and a total of 8 columns including the 6 target features which are malignant,highly_malignant,rude,threat,abuse and loathe.

As seen in the non-null count, we have that all the features including the 6 target features have no missing values as all the features have equal number of instances as the number of cases or entries which is equal to 159571.

We can also see, as mentioned in the previous code, we have 2 features (not target ones) which are data of type object. And the remaining ones, the target features, are numerical integer type values.

- Mathematical/ Analytical Modeling of the Problem

  Describe the mathematical, statistical and analytics modelling done during this project along with the proper justification.

[9]:
```
#descriptive quick summary of training dataset:
print('Train data descriptive quick analysis',train.describe())
```

```
Train data descriptive quick analysis            malignant  highly_malignant          rude         threat  \
count  159571.000000     159571.000000  159571.000000  159571.000000
mean        0.095844          0.009996       0.052948       0.002996
std         0.294379          0.099477       0.223931       0.054650
min         0.000000          0.000000       0.000000       0.000000
25%         0.000000          0.000000       0.000000       0.000000
50%         0.000000          0.000000       0.000000       0.000000
75%         0.000000          0.000000       0.000000       0.000000
max         1.000000          1.000000       1.000000       1.000000

               abuse         loathe
count  159571.000000  159571.000000
mean        0.049364       0.008805
std         0.216627       0.093420
min         0.000000       0.000000
25%         0.000000       0.000000
50%         0.000000       0.000000
75%         0.000000       0.000000
max         1.000000       1.000000
```

[116]:
```
train.describe()
```

[116]:

	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

When describing the training dataset, we see we have target columns as numerical values, as seen in the dtyes previously, where we can highlight the following:

- there are no missing values as all the columns have same number of instances as the total number of cases.

- checking the mean and median, we can see all the mean are higher than median, which means all the numerical target features have a right skewed distribution.

- when checking the standard distribution, malignant, rude and abuse have higher deviation than other numerical target features.

- And regarding the maximum, we see a clear preview of outliers.

Here you can see and observe that the means are very small. It gives us a hint that maybe some cases have no label of any target categories on it. Let's study the labels and its cases!

```
[99]: # calculating total count of each category comments. Calculating number of comments in each category
counts = []
categories = list(train.columns.values)
for i in categories:
    counts.append((i, train[i].sum()))
df_stats = pd.DataFrame(counts, columns=['category', 'count'])
df_stats
```

[99]:

	category	count
0	malignant	15294
1	highly_malignant	1595
2	rude	8449
3	threat	478
4	abuse	7877
5	loathe	1405

As can be seen in the above table, malignant is the only target feature that has the highest number of comments and threat is the target category which has the lowest number of cases of comments.

```
]: #descriptive quick summary of testing dataset:
print('Test data descriptive quick analysis',test.describe())
```

```
Test data descriptive quick analysis                     id                                comment_text
count             153164                                 153164
unique            153164                                 153164
top     00001cee341fdb12  Yo bitch Ja Rule is more succesful then you'll...
freq                   1                                      1
```

As of testing, both features set are of object type fetaures, neither mean, std, medium nor maximum are statistically significant.

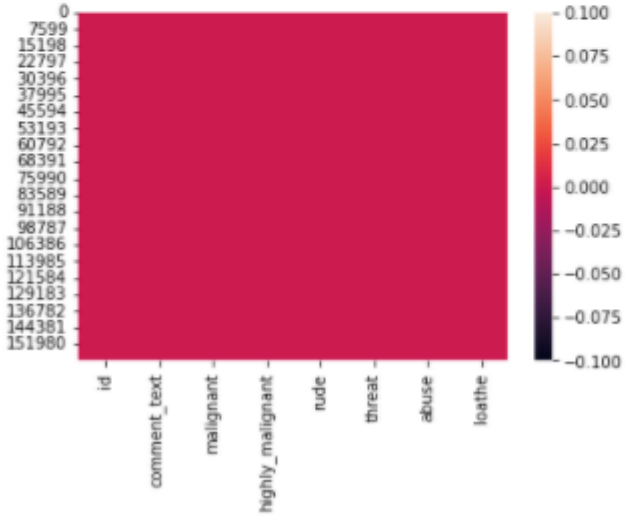
Here all we can highlight is that we have no missing values and the top comment that appears the most is comment id 00001cee341fdb12 which is “Yo bitch Ja Rule is more succesful then you'll...”

```
[11]: # checking null values
print(train.isnull().sum())
```

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

```
[12]: # checking null values through heatmap
print(sns.heatmap(train.isnull()))
```

```
AxesSubplot(0.125,0.125;0.62x0.755)
```

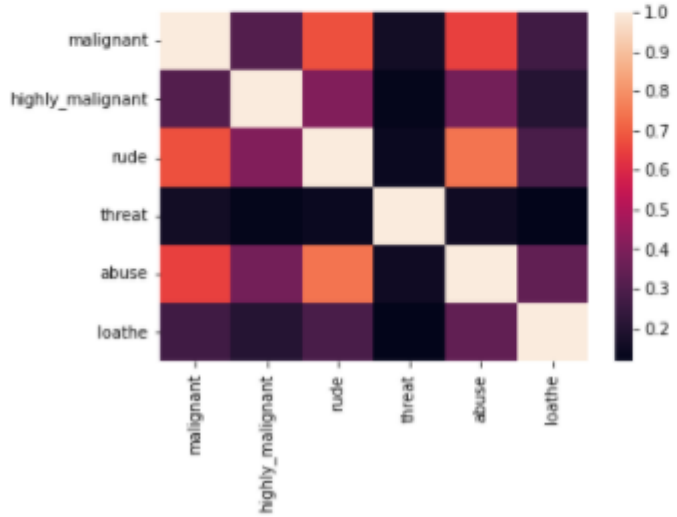


As we said, we can see in the above heatmap no missing values found in the training dataset.

Next, let's examine the correlations among the target variables.

[13]:
```
## checking correlation in dataset
print(train.corr())
print(sns.heatmap(train.corr()))
```

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

                 loathe
malignant        0.266009
highly_malignant 0.201600
rude             0.286867
threat           0.115128
abuse            0.337736
loathe           1.000000
AxesSubplot(0.125,0.125;0.62x0.755)
```



[14]:
```
#Correlation using heatmap:
import matplotlib.pyplot as plt
plt.figure(figsize=(15,7))
sns.heatmap(train.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')
```

[14]: <AxesSubplot:>

As seen in the correlation matrix, there is a high chance of

- rude comments to be abusing and malignant.

- malignant to be rude and abusing.

- abuse comment to be rude and malignant.

We can also highlight that the least correlated target feature with others is threat category.

Checking skewness of the target numerical features:

```
[15]: # checking the skewness for the features:
      train.skew()

      /srv/conda/envs/notebook/lib/python3.7/site-packages
      s deprecated; in a future version this will raise Ty

[15]: malignant            2.745854
      highly_malignant     9.851722
      rude                 3.992817
      threat              18.189001
      abuse                4.160540
      loathe              10.515923
      dtype: float64
```

We see high skewness in our data. So, we need to apply some transformation techique in order to reduce the skewness the data. We apply it later.

And when checking proportion of unlabelled comments is 89% as shown in the following code snapshoot:

```
[145]: unlabelled_in_all = train_df[(train_df['malignant']!=1) & (train_df['highly_malignant']!=1) & (train_df['rude']!=1) &
                                    (train_df['threat']!=1) & (train_df['abuse']!=1) & (train_df['loathe']!=1)]
       print('Percentage of unlabelled comments is ', len(unlabelled_in_all)/len(train_df)*100)

       Percentage of unlabelled comments is  89.83211235124176
```

- Data Preprocessing Done

What were the steps followed for the cleaning of the data? What were the assumptions done and what were the next actions steps over that?

Since we have cases of multilabel data, let's now check that frequency of occurrence of multilabelled data. For that, we need to balance and equilibrized the common dtype of all features:

[17]:
```
df=train
```

Now let's check that frequency of occurence of multilabelled data. For that, we need to balance and equilibrized the common dtype of all features:

[18]:
```
#Convert the comment feature to a NumPy array:
cmt = df['comment_text']
print(cmt.head())

cmt=cmt.to_numpy()
```

```
0    Explanation\nWhy the edits made under my usern...
1    D'aww! He matches this background colour I'm s...
2    Hey man, I'm really not trying to edit war. It...
3    "\nMore\nI can't make any real suggestions on ...
4    You, sir, are my hero. Any chance you remember...
Name: comment_text, dtype: object
```

[19]:
```
#Convert the target features to a NumPy array establizing the common dtype to all corresponding features.
lbl = df[['malignant', 'highly_malignant', 'rude', 'threat', 'abuse', 'loathe']]
print(lbl.head())
lbl = lbl.to_numpy()
```

	malignant	highly_malignant	rude	threat	abuse	loathe
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

checking frequency of occurence of multilabelled data

- ct1 counts samples having atleast one label
- ct2 counts samples having 2 or more than 2 labels
- ct3 counts samples having 3 or more than 3 labels
- ct4 counts samples having 4 or more than 4 labels
- ct5 counts samples having 5 or more than 5 labels
- ct6 counts samples having 6 or more than 6 labels
- ct7 counts samples having 7 or more than 7 labels
- ct8 counts samples having 8 or more than 8 labels

[20]:
```
ct1,ct2,ct3,ct4,ct5,ct6,ct7,ct8 = 0,0,0,0,0,0,0,0
for i in range(lbl.shape[0]):
    ct = np.count_nonzero(lbl[i])
    if ct :
        ct1 = ct1+1
    if ct>1 :
        ct2 = ct2+1
    if ct>2 :
        ct3 = ct3+1
    if ct>3 :
        ct4 = ct4+1
    if ct>4 :
        ct5 = ct5+1
    if ct>5 :
        ct6 = ct6+1
    if ct>6 :
        ct7 = ct7+1
    if ct>7 :
        ct8 = ct8+1
print(ct1)
print(ct2)
print(ct3)
print(ct4)
print(ct5)
print(ct6)
print(ct7)
```

```
print(ct8)
```

```
16225
9865
6385
2176
416
31
0
0
```

Here we can say that the highest number of comments is for the group that has only one unique label.

And when checking the least number of comments, we can see having 6 categories labels or more.

Now, let's check the length of the comments in order to analize if we can reduce the length of the comment in any way in order to avoid sending all the large comments to the ML:

[21]:
```
#checking the length of each feature:
train['length_comments'] = train['comment_text'].str.len() #no olvides
train.head()
```

[21]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length_comments
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	264
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	112
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	233
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	622
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	67

[22]:
```
#printing the maximum and the average length of feature comments:
print(train['length_comments'].max())
train['length_comments'].mean()
```

```
5000
```

[22]:
```
394.138847284281
```

[23]:
```
#which should match with:
x = [len(cmt[i]) for i in range(cmt.shape[0])]
sum(x)/len(x)
```

[23]:
```
394.138847284281
```

[24]:
```
#Length of comment for example 159570
len(cmt[159570])
```

[24]:
```
189
```

Let's now remove comments with higher words as some very large length comments can be seen in our dataset and these can create serious issue in the training dataset, causing training time to increase and accuracy to decrease. Hence, a threshold of 500 characters will

be created and only comments which have length smaller than 500 will be used further.

We will try reduce our dataset by comment length value 500, which means only comments with length smaller than 500 will be used further:

```
[27]: comments = []  #we will try reduce our dataset by comment length value 500, which means only comments with length smaller than 500 will be used further:
      labels = []

      for x in range(cmt.shape[0]):
          if len(cmt[x])<=500:
              comments.append(cmt[x])
              labels.append(lbl[x])

[28]: labels = np.asarray(labels) #converting in array

[29]: print(len(comments)) #comment feature total row length

      125615

[30]: print(len(labels)) #double-checking

      125615
```

Hence, after removing comments longer than 500 characters, we are still left with more than 125k comments, which seems okay for training purposes.

Feature-engineering

Before fitting models, now we need to break down the sentence into unique words by tokenizing the comments. In the tokenize() function, we remove punctuations and special characters. We then lemmatize the comments and filter out comments with length below 3. Besides lemmatization, we also tried stemming but did not get a better result.

The text preprocessing techniques followed before processing the text data are:

For Preprocessing

- we have the following steps:

Removing Punctuations and other special characters. All the punctuation marks in every comment are removed.

Splitting the comments into individual words

Removing Stop Words

Stemming and Lemmatising. Lemmatisation:Inflected forms ofwordswhichmay be different verb forms or sigular/plural forms etc. are called lemma. For ex. go and gone are inflected forms or lemma of the word, gone. The process of grouping these lemma together is called Lemmatisation. So, Lemmatisation is performed

for every comment.

Applying Count Vectoriser

Splitting dataset into Training and Testing

Preparing a string containing all punctuations to be removed. The string library contains punctuation characters. This is imported and all numbers are appended to this string. Also, we can notice that our comment_text field contains strings such as won't, didn't, etc which contain apostrophe character('). To prevent these words from being converted to wont/didnt, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.

Removal of Stopwords: Frequently occurring common words like articles, prepositions etc. are also called stopwords. So, stopwords are removed for each comment.

maketrans() returns a translation table that maps each character in the punctuation_edit into the character at the same position in the outtab string i.e. it replaces every character in the removal list with a space, since outtab contains a string with spaces.

All these preprocessing are done in the following code:

```
# Convert all comments to lower case
train['comment_text'] = train['comment_text'].str.lower()


# Replace numbers of different types with 'number'
train['comment_text'] = train['comment_text'].str.replace(r'\d+(\.\d+)?', 'number')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'telephonenumber'
train['comment_text'] = train['comment_text'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                'telephonenumber')

# Replace email addresses with 'email'
train['comment_text'] = train['comment_text'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                'email')


# Replace URLs with 'website'
train['comment_text'] = train['comment_text'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                'website')

# Replace money symbols with 'moneysynbols'
train['comment_text'] = train['comment_text'].str.replace(r'£|\$', 'moneysynbols')
```

```
[32]: import re #importing regular expression operations
```

```
[33]: #email-id and website imputation:
train['comment_text'] = train['comment_text'].apply(lambda x: re.sub('b[w.]+?@w+?.w{2,4}b', 'email',x))
#url
train['comment_text'] = train['comment_text'].apply(lambda x:re.sub('(http[s]?S+)|(w+.[A-Za-z]{2,4}S*)', 'website', x))
```

```
[34]: #adding 10 digits to the actual string punctuation:
import string
print(string.punctuation)
punctuation_editing = string.punctuation.replace('\'','') +"0123456789"
print (punctuation_editing)
output = "                                "
trans = str.maketrans(punctuation_editing, output)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
!"#$%&()*+,-./:;<=>?@[\]^_`{|}~0123456789
```

```
[35]: import nltk #from nltk library importing stopwords
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
[35]: True
```

```
[36]: #importing stopwords
from nltk.corpus import stopwords
```

Updating the list of stop words

Stop words are those words that are frequently used in both written and verbal communication and do not have either any impact on our training machine.E.g. is, this, us,etc.

Hence can be directly removed. Hence letters 'u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure, will be added to the list of stop words imported directly:

[36]:
```
#importing stopwords
from nltk.corpus import stopwords
```

Updating the list of stop words

Stop words are those words that are frequently used in both written and verbal communication and do not have either any impact on our training machine.E.g. is, this, us,etc. Hence can be directly removed. Hence letters 'u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure, will be added to the list of stop words imported directly.

[37]:
```
#string punstuation and stopword setting:
train['comment_text'] = train['comment_text'].apply(lambda x: ' '.join(
    unit for unit in x.split() if unit not in punctuation_editing))

stop_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])
stop_words.add('subject')
stop_words.add('http')
train['comment_text'] = train['comment_text'].apply(lambda x: ' '.join(
    unit for unit in x.split() if unit not in stop_words))
```

[38]:
```
#define function remove_stopords:
def remove_stopwords(lines):
    return " ".join([wrd for wrd in str(lines).split() if wrd not in stop_words])
train['comment_text']  = train['comment_text'] .apply(lambda x: remove_stopwords(x))
```

Now the comment text will be smaller because all stopwords will be removed.

[39]:
```
#strip and remove punctuation from a string
import re
train['comment_text'] = train['comment_text'].apply(lambda z: re.sub('[%s]' % re.escape(string.punctuation), '' , z))
```

[40]:
```
##remove words and digits #https://www.analyticsvidhya.com/blog/2021/06/must-known-techniques-for-text-preprocessing-in-nlp/
train['comment_text'] = train['comment_text'].apply(lambda x: re.sub('W*dw*','',x))
```

sys.getrecursionlimit() method is used to find the current recursion limit of the interpreter or to find the maximum depth of the Python interpreter stack. This limit prevents any program from getting into infinite recursion, Otherwise infinite recursion will lead to overflow of the C stack and crash the Python.

[41]:
```
import sys
print(sys.getrecursionlimit())
```

```
3000
```

[42]:
```
sys.setrecursionlimit(40000) #setting recursion limit
```

Stemming and Lemmatizing

Stemming is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word.E.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem". This helps in achieving the training process with a better accuracy.

Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly

identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

Stemming just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling. Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma. Sometimes, the same word can have multiple different Lemmas

The wordnet library in nltk will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

```
[43]: #Stemming
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
'''def stem_words(text):
    while True:
        return ' '.join([stemmer.stem(word) for word in text.split()])
train['comment_text'] = train['comment_text'].apply(lambda x: stem_words(x))'''

train['comment_text'] = train['comment_text'].apply(lambda lines: ' '.join([stemmer.stem(wrd) for wrd in lines.split()]))

[44]: import nltk #importing nltk and downloading wordnet
nltk.download('wordnet')

[nltk_data] Downloading package wordnet to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.

[44]: True

[45]: import nltk #importing nltk and downloading omw 1.4
nltk.download('omw-1.4')

[nltk_data] Downloading package omw-1.4 to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/omw-1.4.zip.

[45]: True

[46]: #appling the function lemmatize_words on the comment text feature
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def lemmatize_words(lines):
    return " ".join([lemmatizer.lemmatize(wrd) for wrd in lines.split()])
train['comment_text'] = train['comment_text'].apply(lambda text: lemmatize_words(text))

'''#applying lemmatizer:
lem=WordNetLemmatizer()
train['comment_text'] = train['comment_text'].apply(lambda x: ' '.join(
 lem.lemmatize(t) for t in x.split()))'''

[46]: "#applying lemmatizer:\nlem=WordNetLemmatizer()\ntrain['comment_text'] = train['comment_text'].apply(lambda x: ' '.join(\n lem.lemmatize(t) for t in x.split()))"
```

[48]:
```
#applying stopword for last time
from stop_words import get_stop_words
stop_words = get_stop_words('english')
stop_words.append('')

for x in range(ord('b'), ord('z')+1):
    stop_words.append(chr(x))
```

[49]:
```
print (stop_words)
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any', 'are', "aren't", 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'below', 'betwe
en', 'both', 'but', 'by', "can't", 'cannot', 'could', "couldn't", 'did', "didn't", 'do', 'does', "doesn't", 'doing', "don't", 'down', 'during', 'each', 'few', 'for', 'from',
'further', 'had', "hadn't", 'has', "hasn't", 'have', "haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here', "here's", 'hers', 'herself', 'him', 'himself', 'his',
'how', "how's", 'i', "i'd", "i'll", "i'm", "i've", 'if', 'in', 'into', 'is', "isn't", 'it', "it's", 'its', 'itself', "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself',
'no', 'nor', 'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'same', "shan't", 'she', "she'd", "she'll",
"she's", 'should', "shouldn't", 'so', 'some', 'such', 'than', 'that', "that's", 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', "there's", 'these', 'they',
"they'd", "they'll", "they're", "they've", 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 'very', 'was', "wasn't", 'we', "we'd", "we'll", "we're", "we've",
'were', "weren't", 'what', "what's", 'when', "when's", 'where', "where's", 'which', 'while', 'who', "who's", 'whom', 'why', "why's", 'with', "won't", 'would', "wouldn't", 'yo
u', "you'd", "you'll", "you're", "you've", 'your', 'yours', 'yourself', 'yourselves', '', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

[51]:
```
import nltk #importing nltk and PorterStemmer and WordNetLemmatizer
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

[52]:
```
#create objects for stemmer and lemmatizer and
lemmatiser = WordNetLemmatizer()
stemmer = PorterStemmer()
#download and update words from wordnet library
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /home/jovyan/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

[52]:
```
True
```

[53]:
```
#applying stemmer and lemmatiser on the comments
comments = df['comment_text']
for i in range(len(comments)):
    comments[i] = comments[i].lower().translate(trans)
    l = []
    for wrd in comments[i].split():
        l.append(stemmer.stem(lemmatiser.lemmatize(wrd,pos="v")))
    comments[i] = " ".join(l)
```

Applying Count Vectorizer

Here we can finally convert our comments into a matrix of token counts, which signifies the number of times it occurs.

[54]:
```
#import required library CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer

#create object with CountVectorizer and apply on the comments feature:
cnt_vect = CountVectorizer()
#fitting it to converts comments into bag of words format
tf = cnt_vect.fit_transform(comments)
```

[55]:
```
#Length after having removed all possible stopwords and having done stemmer and lemmatiser and doing the fit_transform through CountVectorizer
train['Cleaned_new_length'] = train.comment_text.str.len() #no olvides
train.head()
```

[55]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe	length_comments	Cleaned_new_length
0	0000997932d777bf	explan eit mae usernam harcor metallica fan we...	0	0	0	0	0	0	264	163
1	000103f0d9cfb60f	aww match backgroun colour im seemingli stick ...	0	0	0	0	0	0	112	100
2	000113f07ec002fd	hey man im realli tri eit war guy constantli r...	0	0	0	0	0	0	233	118
3	0001b41b1c6bb37e	cant make real suggest improv websit section s...	0	0	0	0	0	0	622	332
4	0001d958c54c6e35	you sir hero chanc rememb websit page that on	0	0	0	0	0	0	67	45

[56]:
```
# Total length removal
print ('Original Length', train.length_comments.sum())    #no olvides
print ('New Cleaned Length', train.Cleaned_new_length.sum())
```

```
Original Length 62893130
New Cleaned Length 38055690
```

[62]:
```
#  Convert text into vectors using TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vector = TfidfVectorizer(max_features = 10000, stop_words='english')
x = tf_vector.fit_transform(train['comment_text'])
```

[64]:
```
#training and testing shape:
print(train.shape)
print(test.shape)
```

```
(159571, 10)
(153164, 2)
```

[67]:
```
#train and test split
from sklearn.model_selection import train_test_split, GridSearchCV
y=train['bad']
#Transforming the data to remove the skewness:
from sklearn.preprocessing import power_transform
x=power_transform(X,method='yeo-johnson')
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=56,test_size=.25)
```

[68]:
```
print(y_train.shape)
y_test.shape
```

```
(119678,)
```

[68]:
```
(39893,)
```

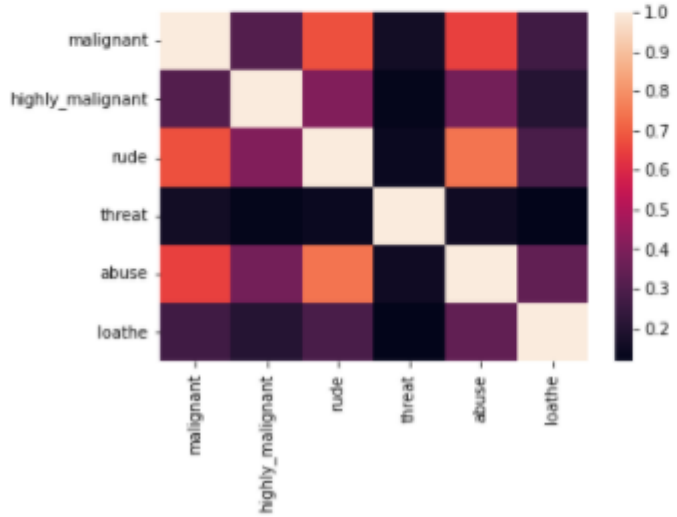
- Data Inputs- Logic- Output Relationships

  In order to describe how the input affects the output, we will use the correlation heatmap to do so:

  Let’s examine the correlations among the target variables.

[13]:
```
## checking correlation in dataset
print(train.corr())
print(sns.heatmap(train.corr()))
```

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

                 loathe
malignant        0.266009
highly_malignant 0.201600
rude             0.286867
threat           0.115128
abuse            0.337736
loathe           1.000000
AxesSubplot(0.125,0.125;0.62x0.755)
```



[14]:
```
#Correlation using heatmap:
import matplotlib.pyplot as plt
plt.figure(figsize=(15,7))
sns.heatmap(train.corr(),annot=True,linewidths=0.5,linecolor="black",fmt='.2f')
```

[14]: <AxesSubplot:>

As seen in the correlation matrix and previously commented, there is a high chance of

- rude comments to be abusing and malignant.

- malignant to be rude and abusing.

- abuse comment to be rude and malignant.

We can also highlight that the least correlated target feature with others is threat category.

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

  The assumption taken by me were:

  - Not exactly an assumption, but yes consideration: Having mean small and near to 0 gave me hint of that maybe some cases has no label of any target categories on it.
  - Not exactly an assumption, but yes consideration: More stopwords we remove, less comment length will go to the training algorithm which is good.
  -

- Hardware and Software Requirements and Tools Used

```
[1]: #installing nltk
     !pip install nltk

Collecting nltk
  Downloading nltk-3.6.7-py3-none-any.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 8.6 MB/s eta 0:00:01
Collecting click
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 18.3 MB/s eta 0:00:01
Requirement already satisfied: joblib in /srv/conda/envs/notebook/lib/python3.7/site-packages (from nltk) (1.0.1)
Collecting tqdm
  Downloading tqdm-4.62.3-py2.py3-none-any.whl (76 kB)
     |████████████████████████████████| 76 kB 8.7 MB/s  eta 0:00:01
Collecting regex>=2021.8.3
  Downloading regex-2021.11.10-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (749 kB)
     |████████████████████████████████| 749 kB 45.0 MB/s eta 0:00:01
Requirement already satisfied: importlib-metadata in /srv/conda/envs/notebook/lib/python3.7/site-packages (from click->nltk) (4.6.3)
Requirement already satisfied: typing-extensions>=3.6.4 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from importlib-metadata->click->nltk) (3.10.0.0)
Requirement already satisfied: zipp>=0.5 in /srv/conda/envs/notebook/lib/python3.7/site-packages (from importlib-metadata->click->nltk) (3.5.0)
Installing collected packages: tqdm, regex, click, nltk
Successfully installed click-8.0.3 nltk-3.6.7 regex-2021.11.10 tqdm-4.62.3

[2]: #importing few of the libraries i will need for next steps:
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     import seaborn as sns
     from nltk.stem import WordNetLemmatizer
     import nltk
     from nltk.corpus import  stopwords
     import string
```

Installed nltk library for natural language processing.

Loaded pandas as it is a software library written for the Python programming language for data manipulation and analysis.

Loaded Numpy as it is library to be used for mathematical operations which we basically used for arrays, transpose, zeros and random choice.

Loaded also matplotlib to plot the data and executed %inline so that the plot is visiplot when executing the code.

[47]:
```
!pip install stop_words #installing stop_words
```

```
Collecting stop_words
  Downloading stop-words-2018.7.23.tar.gz (31 kB)
Building wheels for collected packages: stop-words
  Building wheel for stop-words (setup.py) ... done
  Created wheel for stop-words: filename=stop_words-2018.7.23-py3-none-any.whl size=32916 sha256=ac19f3f2b097db65566bdaf376dca71c5eb3e1d8cdfcce54c002c357522195b0
  Stored in directory: /home/jovyan/.cache/pip/wheels/fb/86/b2/277b10b1ce9f73ce15059bf6975d4547cc4ec3feeb651978e9
Successfully built stop-words
Installing collected packages: stop-words
Successfully installed stop-words-2018.7.23
```

[61]:
```
#Packages Loading for our machine learning algorithms
from nltk.stem.wordnet import WordNetLemmatizer
from timeit import default_timer as timer

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import fbeta_score
from statistics import mean
from sklearn.metrics import hamming_loss
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve

from sklearn.metrics import roc_auc_score, confusion_matrix
import statistics
from sklearn.metrics import recall_score

from wordcloud import WordCloud
from collections import Counter

from sklearn.pipeline import Pipeline

from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import VotingClassifier
import xgboost as xgb
import warnings
```

Imported Stopwords for text preprocessing.

Lemmatization is used for the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form.

Seaborn for visualization.

TfidfVectorizer for conversion of a collection of raw documents to a matrix of TF-IDF features.

Loaded few classification algorithms such as LogisticRegression, MultinomialNB, LinearSVC, RandomForestClassifier, DecisiontreeClassifier, AdaboostClassifier, BaggingClassifier, GradientBoostingClassifier, VotingClassifier.

For results evaluation and comparison, loaded several functions from sklearn.metrics such as accuracy_score, f1_score, recall_score, precision_score, roc_auc_score, roc_curve, confusion_matrix, cross_val_score, hamming_loss.

For model training, loaded few function such as StratifiedKFold, GridSearchCV, ShuffleSplit, Learning_curve.

Also loaded WordCloud for visualization of highlighted words in our text that are the most frequent.

Also loaded Counter function from library collections.

```
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

Loaded XGBoost just to try if we can train it the lart data we have. Training it took longer than expected, so finally, decided not to get its results.

Loaded also warning in order to avoid extra warnings messages when executing code.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  Describe the approaches you followed, both statistical and analytical, for solving of this problem:

  Basically, we tried to clean the comments as much as possible, understand the relations and correlation between each feature and classify it in the 6 targets variables options we have through several classification algorithms.

  Compared the means and statistical deviation in order to get some views on the distribution and also did some plotting to see visually the distribution of the data.

  And mostly importantly, we pre-processed and prepared the data and comments for our machine learning through different techniques of nltk.

  Approaches like cross validation and classification reports are done to analyse the results of algorithm and training performance.

  We used accuracy, precision, recall, f1 score, hamming loss and log loss for validation and results evaluation and comparison.

- Testing of Identified Approaches (Algorithms)

  Listing down all the algorithms used for the training and testing:

  I used and compared the following algorithms:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- AdaBoost Classifier
- KNeighbor Classifier

- Run and Evaluate selected models

  Describe all the algorithms used along with the snapshot of their code and what were the results observed over different evaluation metrics.

  - As we mentioned we used Logistic Regression which is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

```
[69]: #Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score, confusion_matrix,classification_report
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)

LG.fit(x_train, y_train)

y_pred_train = LG.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = LG.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9585387456341182
Test accuracy is 0.9562579901235806
[[35644   205]
 [ 1540  2504]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     35849
           1       0.92      0.62      0.74      4044

    accuracy                           0.96     39893
   macro avg       0.94      0.81      0.86     39893
weighted avg       0.96      0.96      0.95     39893
```

- Decision tree works by successively splitting the dataset into small segments until the target variable are the same or until the dataset can no longer be split. It's a greedy algorithm which make the best decision at the given time without concern for the global optimality

The construction is similar:

1. Assign all training instances to the root of the tree. Set current node to root node.
2. Find the split feature and split value based on the split criterion such as information gain, information gain ratio or gini coefficient.
3. Partition all data instances at the node based on the split feature and threshold value.
4. Denote each partition as a child node of the current node.
5. For each child node:
    1. If the child node is "pure" (has instances from only one class), tag it as a leaf and return.
    2. Else, set the child node as the current node and recurse to step 2.

[70]:
```
# DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
y_pred_train = DT.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = DT.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

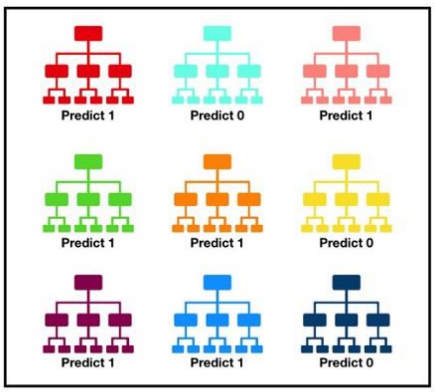
```
Training accuracy is 0.9987215695449456
Test accuracy is 0.9389867896623468
[[34680  1169]
 [ 1265  2779]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     35849
           1       0.70      0.69      0.70      4044

    accuracy                           0.94     39893
   macro avg       0.83      0.83      0.83     39893
weighted avg       0.94      0.94      0.94     39893
```

[71]:
```
#RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9987048580357292
Test accuracy is 0.9560073195798762
[[35475   374]
 [ 1381  2663]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     35849
           1       0.88      0.66      0.75      4044

    accuracy                           0.96     39893
   macro avg       0.92      0.82      0.86     39893
weighted avg       0.95      0.96      0.95     39893
```

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).



Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple: wisdom of crowds. The reason is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all error don't occur in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

[72]:
```
#AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=100)
ada.fit(x_train, y_train)
y_pred_train = ada.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = ada.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9501495680074867
Test accuracy is 0.9503421652921565
[[35517   332]
 [ 1649  2395]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     35849
           1       0.88      0.59      0.71      4044

    accuracy                           0.95     39893
   macro avg       0.92      0.79      0.84     39893
weighted avg       0.95      0.95      0.95     39893
```

An AdaBoost Classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

AdaBoost algorithm, called Adaptive Boosting, as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances. Boosting is used to reduce bias as well as variance for supervised learning. It works on the principle of learners growing sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones.

```
[73]: from sklearn.neighbors import KNeighborsClassifier

[74]: #KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
y_pred_train = knn.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = knn.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9238707197647019
Test accuracy is 0.9199859624495526
[[35698   151]
 [ 3041  1003]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     35849
           1       0.87      0.25      0.39      4044

    accuracy                           0.92     39893
   macro avg       0.90      0.62      0.67     39893
weighted avg       0.92      0.92      0.90     39893
```

K-Nearest-Neighbor is a Machine Learning supervised type instance-based algorithm. It can be used to classify new samples (discrete values) or to predict (regression, continuous values). Being a simple method, it is ideal to enter the world of Machine Learning. It essentially serves to classify values by looking for the "most similar" data points (by proximity) learned in the training stage and making guesses for new points based on that classification.

After analysing and comparing the results, we decided to go with Random Forest algorithm as it gave us the best results such as accuracy, precision,recall and therefore f1Score.

- Key Metrics for success in solving problem under consideration

  The key metrics I used in order to compare the results were Training and Testing accuracy, Hamming Loss and Logg Loss. I also used and compared cross validation score alongside the precision, the recall score and the f1 score as shown in below example:

[75]:
```
# RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
```

```
Training accuracy is 0.9986965022811211
Test accuracy is 0.9559822525255057
```

[69]:
```
from sklearn.metrics import hamming_loss       #we will evaluate our ML with log loss, hamming_loss, Recall and Precision.
from sklearn.metrics import log_loss

def evaluate_score(Y_test,predict):
    loss = hamming_loss(Y_test,predict)
    print("Hamming_loss : {}".format(loss*100))
    try :
        loss = log_loss(Y_test,predict)
    except :
        loss = log_loss(Y_test,predict.toarray())
    print("Log_loss : {}".format(loss))
```

[70]:
```
# calculate results
evaluate_score(y_test,y_pred_train[0:39893])
```

```
Hamming_loss : 17.486777128819593
Log_loss : 6.039788502543476
```

[76]:
```
#cross validation of Random forest
cvs=cross_val_score(RF, x, y, cv=2, scoring='accuracy')
```

[77]:
```
#mwan of cross validation:
cvmean=cvs.mean()
print('cross validation score :',cvmean*100)
```

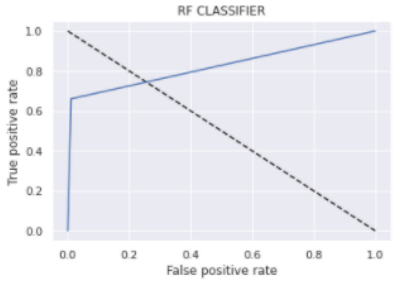
```
cross validation score : 95.44152747001438
```

[78]:
```
#confusion matrix and classification report:
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
[[35466   383]
 [ 1373  2671]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     35849
           1       0.87      0.66      0.75      4044

    accuracy                           0.96     39893
   macro avg       0.92      0.82      0.86     39893
weighted avg       0.95      0.96      0.95     39893
```

I also did the ROC and AUC curve plot for the selected algorithm:

[79]:
```
#Plotting the graph of AUC
# model is performing good :
from sklearn.metrics import roc_curve, auc
fpr,tpr,thresholds=roc_curve(y_test,y_pred_test)
roc_auc=auc(fpr,tpr)
plt.plot([0,1],[1,0],'k--')
plt.plot(fpr,tpr,label = 'RF Classifier')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('RF CLASSIFIER')
plt.show()
#more the area under curve, more will be the better prediction
```



Then I also checked weights of the selected RandomForest algorithm and checked the words which have more weights and hence makes a comment to have more chance to be malignent:

[81]:
```
import eli5
eli5.show_weights(RF,vec = tf_vector, top = 15)  #random forest
# top 15 words  which makes a comment malignent
```

[81]:

Weight	Feature
0.1082 ± 0.0838	fuck
0.0275 ± 0.0286	shit
0.0226 ± 0.0202	suck
0.0208 ± 0.0126	iiot
0.0191 ± 0.0144	stupi
0.0177 ± 0.0191	bitch
0.0169 ± 0.0148	asshol
0.0129 ± 0.0113	faggot
0.0120 ± 0.0122	cunt
0.0119 ± 0.0127	ick
0.0103 ± 0.0062	gay
0.0087 ± 0.0027	websit
0.0080 ± 0.0066	hell
0.0076 ± 0.0057	moron
0.0070 ± 0.0060	bastar
… 9985 more …	

[82]:
```
#fit the test dataset
test_data =tf_vector.fit_transform(test['comment_text'])
test_data
```

[82]:
```
<153164x10000 sparse matrix of type '<class 'numpy.float64'>'
        with 2940344 stored elements in Compressed Sparse Row format>
```

[83]:
```
#Random Forest prediciton:
prediction=RF.predict(test_data)
prediction
```

[83]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

[84]:
```
#importing joblib for algorithm saving:
import joblib
joblib.dump(RF,"malig.pkl")
```

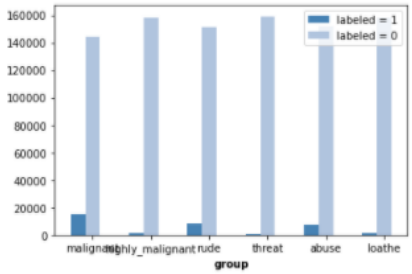
[84]:
```
['malig.pkl']
```

Once checked the weights of the words that makes a comment malignent, we fitted the tf_vector on the test feature “comment_text” and checked and predicted on test data.
Then finally, we saved the model as “malig.pkl” so that the technical engineer can implement wherever is needed.

- Visualizations

  Mention all the plots made along with their pictures and what were the inferences and observations obtained from those.

```
[3]: # Code to draw bar graph for visualising distribution of classes within each label.
barWidth = 0.25

bars1 = [sum(train['malignant'] == 1), sum(train['highly_malignant'] == 1), sum(train['rude'] == 1), sum(train['threat'] == 1),
         sum(train['abuse'] == 1), sum(train['loathe'] == 1)]
bars2 = [sum(train['malignant'] == 0), sum(train['highly_malignant'] == 0), sum(train['rude'] == 0), sum(train['threat'] == 0),
         sum(train['abuse'] == 0), sum(train['loathe'] == 0)]

r1 = np.arange(len(bars1))
r2 = [x + barWidth for x in r1]

plt.bar(r1, bars1, color='steelblue', width=barWidth, label='labeled = 1')
plt.bar(r2, bars2, color='lightsteelblue', width=barWidth, label='labeled = 0')

plt.xlabel('group', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(bars1))], ['malignant', 'highly_malignant', 'rude', 'threat', 'abuse',
                                                       'loathe'])
plt.legend()
plt.show()
```

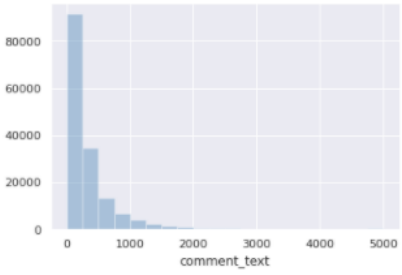


In the above plot, at first sight we can notice we have way more cases of non labelled cases than labelled cases.

And when checking and doing the unique categorization of each comment, we see the malignant, highly malignant and abuse have more comments than rude, abuse and loathe.

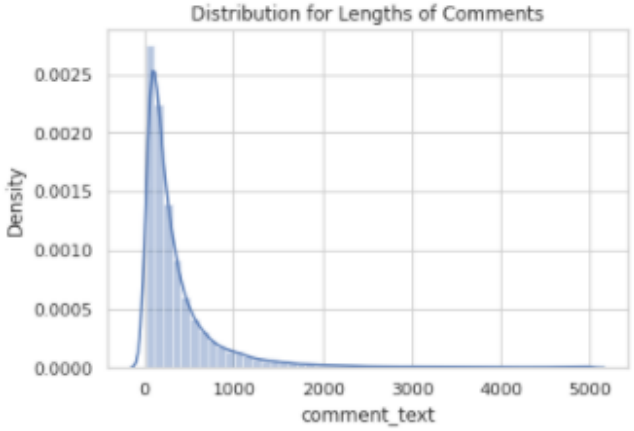
```
[118]: #Below is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words.
sns.set(color_codes=True)
comment_len = train.comment_text.str.len()
sns.distplot(comment_len, kde=False, bins=20, color="steelblue")
```

```
[118]: <AxesSubplot:xlabel='comment_text'>
```



#comment below

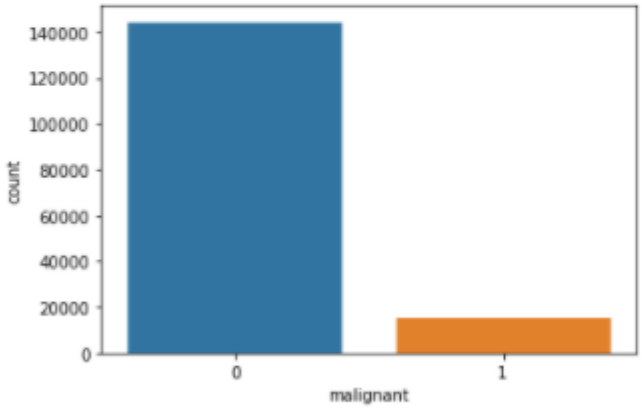
```
[102]: lens = train.comment_text.str.len()
       sns.distplot(lens)
       plt.title("Distribution for Lengths of Comments")
       plt.show()
```



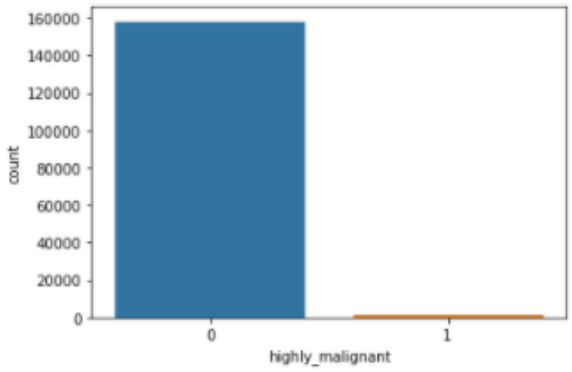
Most of the comment text length are within 1500 characters, with some up to 5,000 characters long.

[16]:
```
#plotting the count values for each target feature:
col=['malignant','highly_malignant','loathe','rude','abuse','threat']
for i in col:
    print(i)
    print("\n")
    print(train[i].value_counts())
    sns.countplot(train[i])
    plt.show()
```

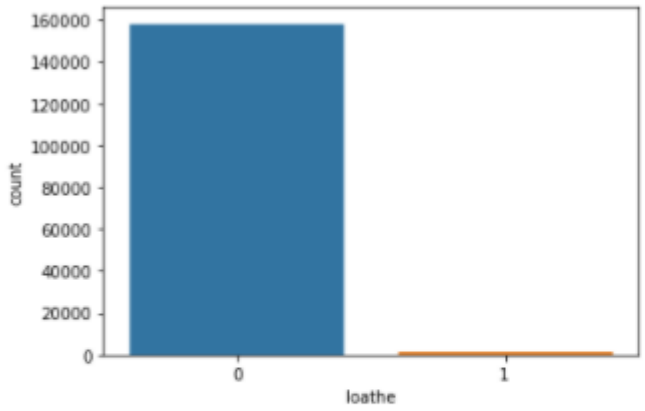
```
/srv/conda/envs/notebook/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning
lid positional argument will be `data`, and passing other arguments without an explicit keywo
  FutureWarning
malignant


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




```
highly_malignant


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

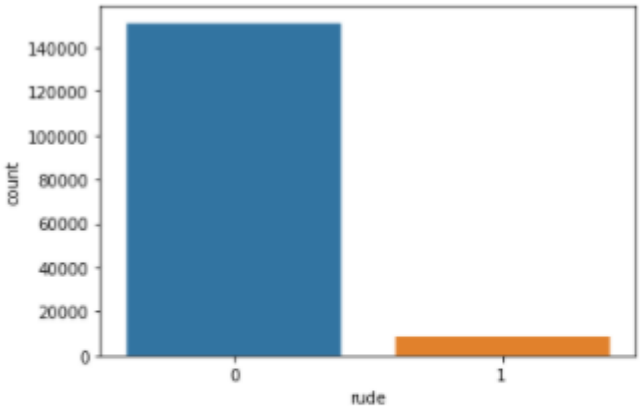



```
loathe


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



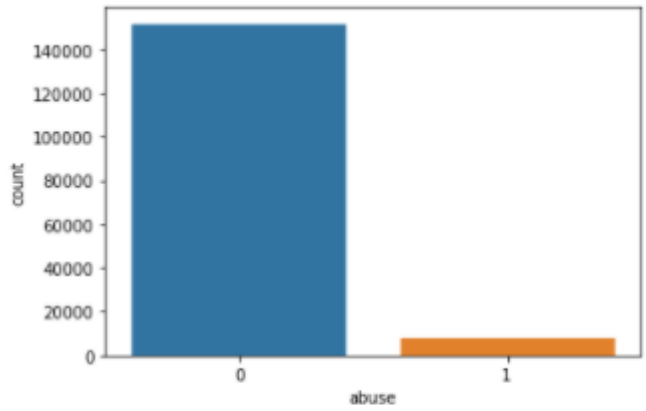

```
rude
```

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



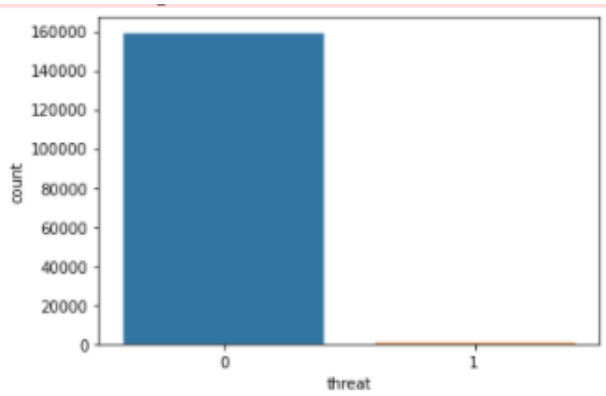

```
abuse
```

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




```
threat
```

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



As we said, we have way more cases of unlabelled comments than comments that have at least one category defined.

So let's check the comments proportion that we have as unlabelled comments:

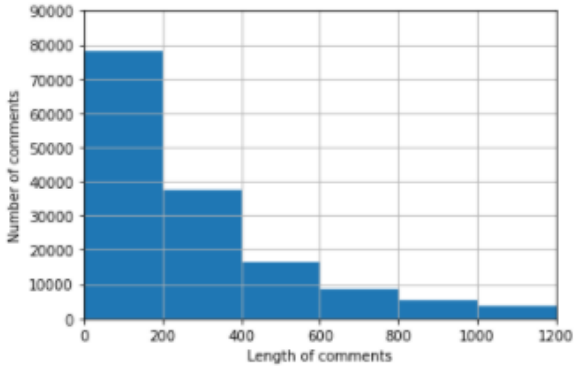
```
unlabelled_in_all = train_df[(train_df['malignant']!=1) & (train_df['highly_malignant']!=1) & (train_df['rude']!=1) &
                             (train_df['threat']!=1) & (train_df['abuse']!=1) & (train_df['loathe']!=1)]
print('Percentage of unlabelled comments is ', len(unlabelled_in_all)/len(train_df)*100)
```

```
Percentage of unlabelled comments is  89.83211235124176
```

After that, I also checked length of comments and compared it with its number of comments having same length:

```
[25]: # average length of feature comments and number of comments for each length of bin of comments:
x = [len(cmt[i]) for i in range(cmt.shape[0])]

print('average length of comment: {:.3f}'.format(sum(x)/len(x)) )
bins = [1,200,400,600,800,1000,1200]
plt.hist(x, bins=bins)
plt.xlabel('Length of comments')
plt.ylabel('Number of comments')
plt.axis([0, 1200, 0, 90000])
plt.grid(True)
plt.show()
```

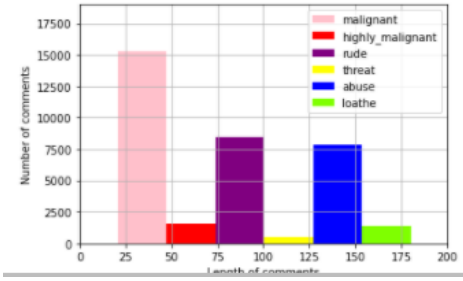
```
average length of comment: 394.139
```



As we commented in the previous distribution plots, here we can see the that length of comments are more likely to be between 0 and 1200 as we mentioned previously.

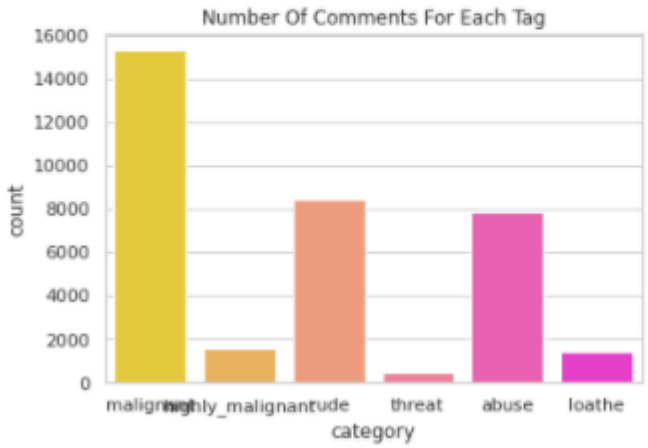
Now let's see number of comments for each length of comments:

```
z = np.zeros(lbl.shape)
for xx in range(cmt.shape[0]):
    l = len(cmt[xx])
    if lbl[xx][0] :
        z[xx][0] = 1
    if lbl[xx][1] :
        z[xx][1] = 1
    if lbl[xx][2] :
        z[xx][2] = 1
    if lbl[xx][3] :
        z[xx][3] = 1
    if lbl[xx][4] :
        z[xx][4] = 1
    if lbl[xx][5] :
        z[xx][5] = 1

labelsplt = ['malignant','highly_malignant','rude', 'threat', 'abuse','loathe']
color = ['pink','red','purple','yellow','blue','chartreuse']
plt.hist(lbl,bins = bins,label = labelsplt,color = color)
plt.axis([0, 200, 0, 19000])
plt.xlabel('Length of comments')
plt.ylabel('Number of comments')
plt.legend()
plt.grid(True)
plt.show()
```



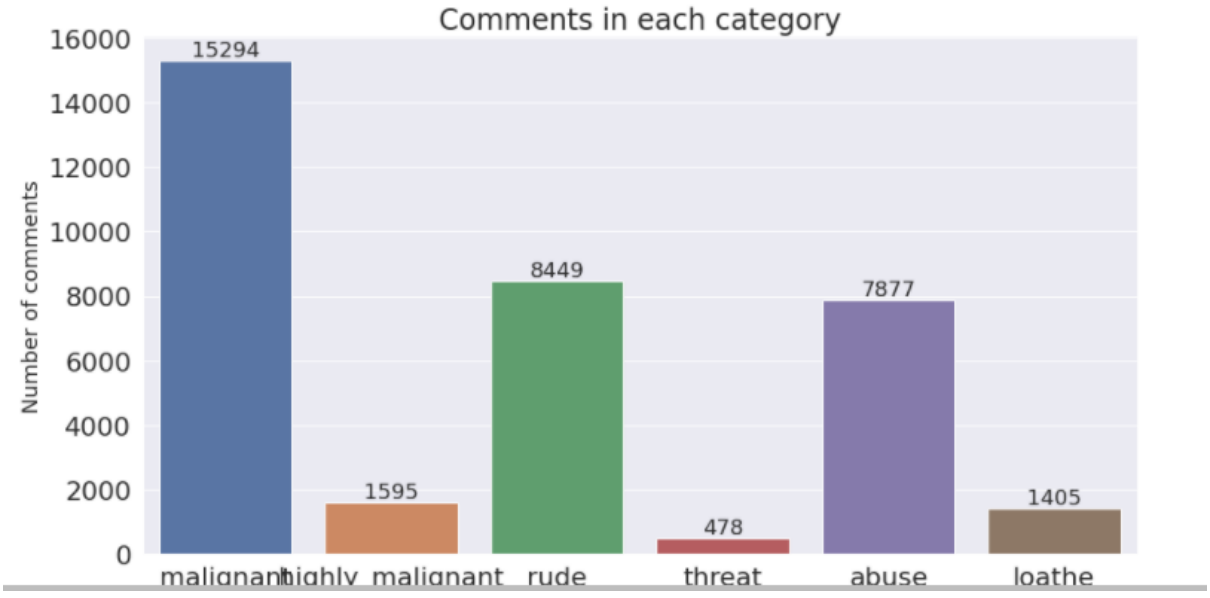
#comment below

```
[104]: # df_toxic.sum().plot(kind="bar")

sns.set(style="whitegrid")
sns.barplot(x='category', y='count', data=df_stats, palette="spring_r")
plt.title("Number Of Comments For Each Tag")
plt.show()
```



#comment below

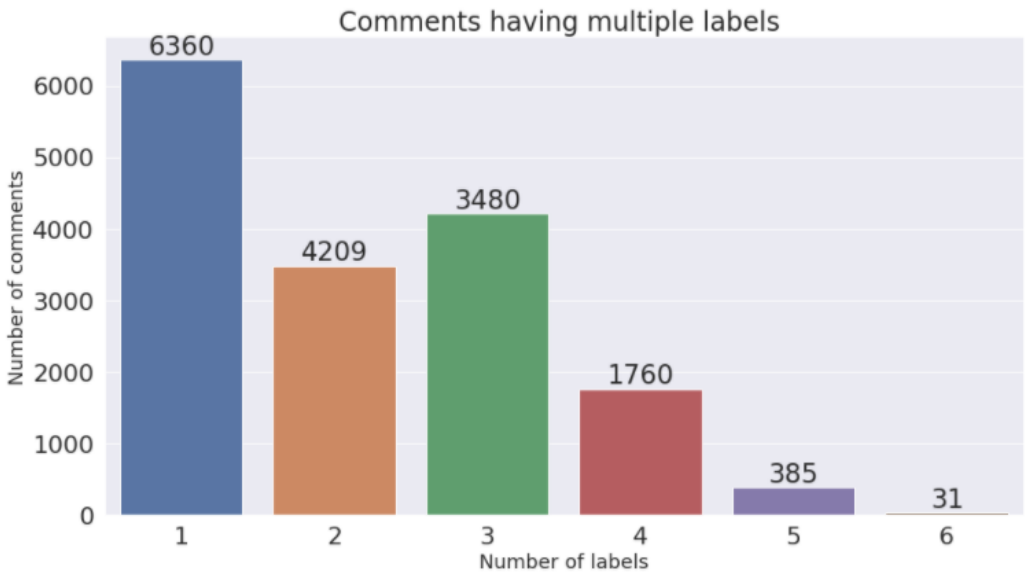
```
plt.xlabel('Comment Type ', fontsize=18)

#adding the text labels
rects = ax.patches
labels = data_raw.iloc[:,2:].sum().values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom', fontsize=18)

plt.show()
```



In the above 3 plots, we could see that malignant, rude and abuse are the 3 categories having the highest number of comments.

And high-malignant, threat and loathe are the 3 categories with lowest number of comments cases.

```
#1.3. Calculating number of comments having multiple labels
rowSums = data_raw.iloc[:,2:].sum(axis=1)
multiLabel_counts = rowSums.value_counts()
multiLabel_counts = multiLabel_counts.iloc[1:]

sns.set(font_scale = 2)
plt.figure(figsize=(15,8))

ax = sns.barplot(multiLabel_counts.index, multiLabel_counts.values)

plt.title("Comments having multiple labels ")
plt.ylabel('Number of comments', fontsize=18)
plt.xlabel('Number of labels', fontsize=18)

#adding the text labels
rects = ax.patches
labels = multiLabel_counts.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')

plt.show()
```



In the above plot, we can see the highest number of comments is for the group that has only one unique label and lowest number of comments is for the group that has 6 or more than 6 multilabel.

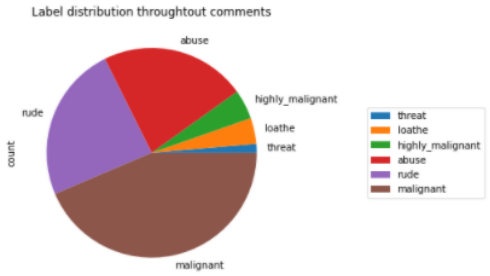
Now, let's check the words that were and contributed most to the different labels through the word cloud plot:

```
[58]: #In order to get an idea of what are the words that contribute the most to different labels, we generate word clouds.
#Getting words which are highly insulting and ofensive and are found more often in our dataset:
from wordcloud import WordCloud
cloud = WordCloud(width=600,height=400,background_color='black',max_words=50).generate(' '.join(train['comment_text'][train['malignant']==1]))
plt.figure(figsize=(10,8),facecolor='k')
plt.imshow(cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Here above you can see the words with the largest dimension has the highest influence on the label of the comments.

```
[59]: #Target label distribution over and in the comments:
columns_target = ['malignant','highly_malignant','rude','threat','abuse','loathe']
df_dist= train[columns_target].sum()\
                              .to_frame()\
                              .rename(columns={0: 'count'})\
                              .sort_values('count')

df_dist.plot.pie(y='count',title='Label distribution throughtout comments',figsize=(5, 5))\
                              .legend(loc='center left', bbox_to_anchor=(1.3, 0.5))
```

```
[59]: <matplotlib.legend.Legend at 0x7f2cb7031d90>
```

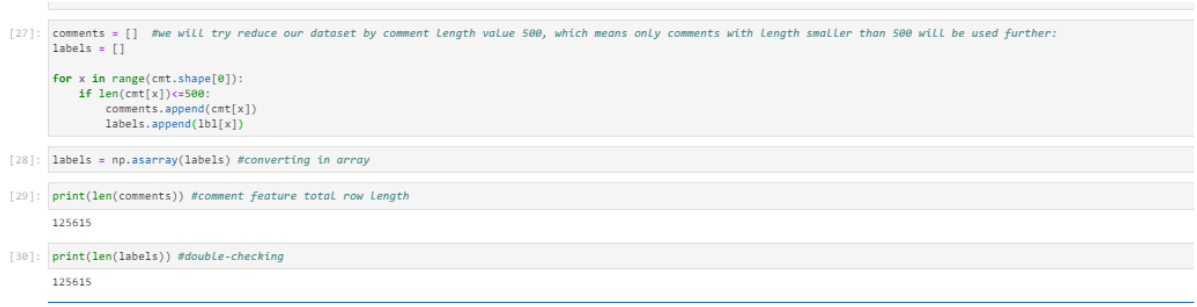


As we said previously, malignant category has the highest number of comments followed by rude and abuse which have the second and third place when comparing the list of highest number of comments in the dataset.

And the second place goes to having 3 labels at the same time as the second highest number of comments.

And when checking the least number of comments, we can see having 6 categories labels.

Let's now remove comments with higher words as some very large length comments can be seen in our dataset and these can create serious issue in the training dataset, causing training time to increase and accuracy to decrease. Hence, a threshold of 500 characters will be created and only comments which have length smaller than 500 will be used further.



```
[27]: comments = []  #we will try reduce our dataset by comment length value 500, which means only comments with length smaller than 500 will be used further:
      labels = []

      for x in range(cmt.shape[0]):
          if len(cmt[x])<=500:
              comments.append(cmt[x])
              labels.append(lbl[x])

[28]: labels = np.asarray(labels) #converting in array

[29]: print(len(comments)) #comment feature total row length
      125615

[30]: print(len(labels)) #double-checking
      125615
```

- ## Interpretation of the Results

  Having malignant, highly_malignant, rude, threat, abuse and/or loathe as multilabel of the target variable, we noticed we had large dataset that includes comments in alphanumeric format which we need to process through nltk library before we add them in our machine learning training and testing.

  We noticed that the training data contains 159,571 observations with 8 columns and the test data contains 153,164 observations with 2 columns.

As we see, the comment id and comment itself are the 2 features whose data type is object. The remaining ones, the target columns, are the numerical ones.

As seen in the non-null count, we have that all the features including the 6 target features have no missing values as all the features have equal number of instances as the number of cases or entries which is equal to 159571.

From the statistic description we saw:

- saw all the mean are higher than median, which means all the numerical target features have a right skewed distribution.

- regarding standard distribution: malignant, rude and abuse have higher deviation than other numerical target features.

- and regarding the maximum, we see a clear preview of outliers.

We also analysed that malignant is the only target feature that has the highest number of comments and threat is the target category which has the lowest number of cases of comments.

As seen in the correlation matrix, there is a high chance of:

- rude comments to be abusing and malignant.

- malignant to be rude and abusing.

- abuse comment to be rude and malignant.

We can also highlight that the least correlated target feature with others is threat category.

And when checking proportion of unlabelled comments, this is 89%.

When checking the occurrence of multilabelled data, we saw that the highest number of comments is for the group that has only one unique label. And when checking the least number of comments, we can see this is when having 6 categories labels or more.

As you may have seen, we removed comments with higher words as some very large length comments could be seen create serious issue in the training dataset, causing training time to increase and accuracy to decrease. Hence, a threshold of 500 characters was put and only comments which have length smaller than 500 was used for our training purposes.

For Feature-engineering, we tokenized the comments. In the tokenize function, we remove punctuations and special characters. We then lemmatize the comments and besides, we also tried stemming. And applied Count Vectoriser where we can finally converted our comments into a matrix of token counts. After preprocessing, dealing and removing unnecessary stopwords and after using maketrans, the comment text was smaller because all stopwords were removed.

Summarizing EDA, we can say:

Basically, we tried to clean the comments as much as possible, understand the relations and correlation between each feature and classify it in the 6 targets variables options we have through several classification algorithms.

Compared the means and statistical deviation in order to get some views on the distribution and also did some plotting to see visually the distribution of the data.

And mostly importantly, we pre-processed and prepared the data and comments for our machine learning through different techniques of nltk.

Approaches like cross validation and classification reports are done to analyse the results of algorithm and training performance.

We used accuracy, precision, recall, f1 score, hamming loss and log loss for validation and results evaluation and comparison.

Throughout the visualizations, we saw the following results or findings:

- at first sight we can notice we have way more cases of non labelled cases than labelled cases.
- and when checking and doing the unique categorization of each comment, we see the malignant, highly malignant and abuse have more comments than rude, abuse and loathe.
- most of the comment text length are within 1500 characters, with some up to 5,000 characters long.
- we could see that malignant, rude and abuse are the 3 categories having the highest number of comments.
- and high-malignant, threat and loathe are the 3 categories with lowest number of comments cases.
- we can see the highest number of comments is for the group that has only one unique label and lowest number of comments is for the group that has 6 or more than 6 multilabel. Which means, the higher the number of multilabel, less are the instances that occur.
- As we said previously, malignant category has the highest number of comments followed by rude and abuse which have the second and third place when comparing the list of highest number of comments in the dataset.
- And the second place goes to having 3 labels at the same time as the second highest number of comments.

- And when checking the least number of comments, we can see having 6 categories labels.
- Throughout the word cloud plot, we could check the words that were and contributed most to the different labels which for example are websit, nigger,moron, faggot, bark, pig pig, bark bark, shit shit, suck suck, f*k f*k…

And finally comparing the algorithms which is the one important part of the study has the following results:

The algorithms I used for the training and testing was :

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- AdaBoost Classifier
- KNeighbor Classifier

[69]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score, confusion_matrix,classification_report
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)

LG.fit(x_train, y_train)

y_pred_train = LG.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = LG.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9585387456341182
Test accuracy is 0.9562579901235806
[[35644   205]
 [ 1540  2504]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     35849
           1       0.92      0.62      0.74      4044

    accuracy                           0.96     39893
   macro avg       0.94      0.81      0.86     39893
weighted avg       0.96      0.96      0.95     39893
```

[72]:
```
#AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=100)
ada.fit(x_train, y_train)
y_pred_train = ada.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = ada.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9501495680074867
Test accuracy is 0.9503421652921565
[[35517   332]
 [ 1649  2395]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     35849
           1       0.88      0.59      0.71      4044

    accuracy                           0.95     39893
   macro avg       0.92      0.79      0.84     39893
weighted avg       0.95      0.95      0.95     39893
```

[70]:
```
# DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
y_pred_train = DT.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = DT.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9987215695449456
Test accuracy is 0.9389867896623468
[[34680  1169]
 [ 1265  2779]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     35849
           1       0.70      0.69      0.70      4044

    accuracy                           0.94     39893
   macro avg       0.83      0.83      0.83     39893
weighted avg       0.94      0.94      0.94     39893
```

[71]:
```
#RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9987048580357292
Test accuracy is 0.9560073195798762
[[35475   374]
 [ 1381  2663]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     35849
           1       0.88      0.66      0.75      4044

    accuracy                           0.96     39893
   macro avg       0.92      0.82      0.86     39893
weighted avg       0.95      0.96      0.95     39893
```

After analysing and comparing the results, we decided to go with Random Forest algorithm as it gave us the best results such as accuracy, precision,recall and therefore f1Score.

The key metrics I used in order to compare the results were Training and Testing accuracy, Hamming Loss and Logg Loss. I also used and compared cross validation score alongside the precision, the recall score and the f1 score.

I also did the ROC and AUC curve plot for the selected algorithm.

Then, to complete and add to my results, I also checked weights of the selected RandomForest algorithm and checked the words which have more weights and hence makes a comment to have more chance to be malignent.

Once checked the weights of the words that makes a comment malignent, we fitted the tf_vector on the test feature “comment_text” and checked and predicted on test data.

Then finally, we saved the model as “malig.pkl” so that the technical engineer can implement wherever is needed.

CONCLUSION

- Key Findings and Conclusions of the Study

  The key findings, inferences, observations from the whole problem are the following ones:

  The main objective of this study was to define and select an algorithm which will give us the best results when comparing metrics such as Accuracy, Precision, Recall and F1Score alongside hamming loss and log loss.

  Throughout the EDA and selection of the best algorithm we found the following main findings and conclusions that may summarize this study done on the malignant online comments:

  - our target features have a right skewed distribution.

  - regarding standard distribution: malignant, rude and abuse have higher deviation than other target features. Which means malignant,

rude and abuse are the target labels that differs more from the mean than the other target features.

We also analysed that malignant is the only target feature that has the highest number of comments and threat is the target category which has the lowest number of cases of comments.

As commented, we have that:

- rude comments are likely to be abusing and malignant.

- malignant are likely to be rude and abusing.

- abuse comment are likely to be rude and malignant.

We can also highlight that the least correlated target feature with others is threat category.

When checking the occurrence of multilabelled data, we saw that the highest number of comments is for the group that has only one unique label. And when checking the least number of comments, we can see this is when having 6 categories labels or more.

The main conclusions are then:

1. Most of the comments are offensive and hence malignant.
2. Most of the comments are non labelled.
3. Most of the comments can be classified as having only one label.
4. Compared the algorithms we included in our study, Random Forest is the algorithm that gets the highest overall results than other algorithms.

- Learning Outcomes of the Study in respect of Data Science

The following steps were taken to process the data which helped me learn and understand more the process of natural language preprocessing and more:

→ Our comment_text field contains strings such as won't, didn't, etc. which contain apostrophe character('). To prevent these words from

being converted to wont or didn't, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.

make_trans(intab, outtab) function is used. It returns a translation table that maps each character in the intab into the character at the same position in the outtab string.

→ Updating the list of stop words:

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

→ Stemming and Lemmatizing:

The process of converting inflected/derived words to their word stem or the root form is called stemming. Many similar origin words are converted to the same word e.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem".

On the other hand, Lemmatizing is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

The wordnet library in nltk will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

→ Applying Count Vectorizer:

To convert a string of words into a matrix of words with column headers represented by words and their values signifying the frequency of occurrence of the word Count Vectorizer is used.

Stop words were accepted, convert to lowercase, and regular expression as its parameters. Here, we will be supplying our custom list of stop words created earlier and using lowercase option. Regular expression will have its default value.

- Limitations of this work and Scope for Future Work

What are the limitations of this solution provided, the future scope? What all steps/techniques can be followed to further extend this study and improve the results.

Improvement → Although we have tried quite several parameters in refining my model, there can exist a better model which gives greater accuracy.

When finishing this project, I saw other similar projects and tried to compare the structure of study and results obtained in different projects having the same goal.

When I saw those project and find out there are better algorithms an other options to be used for this type of natural language analysis like :

→For the problem transformation method: The label powerset method with MultinomialNB classifier.

→Other projects also used SVM model while also trying out different classifiers with the Binary Relevance Method.

Future Scope

As this project predicts the type of comment like malignant or high malignant, etc, we can add other data as future scope such as the age of the commentator or derivative feature which can easily sensitize words which are classified as malignant or even handle mistaken words to get better accuracy of the result.

ok