Malignant comments Classifier

Submitted by:

Ekansh Gupta

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.

INTRODUCTION

- ## Business Problem Framing

  Different Social media online platforms allows users to express their views in the form of comments across the globe. In addition to this online platforms also suffers with involvement of unexpected comments which can be described as described as abusive language, aggression, cyberbullying, hatefulness and many others making the platform uncomfortable for most of the users.

  This project is aimed at utilization of machine learning to deal with this problem. The motive is to prepare a model which can identify and tag unexpected comments on social media platforms by categorizing them as hateful , abusive , malignant ,loathe ,threat.

- ## Conceptual Background of the Domain Problem

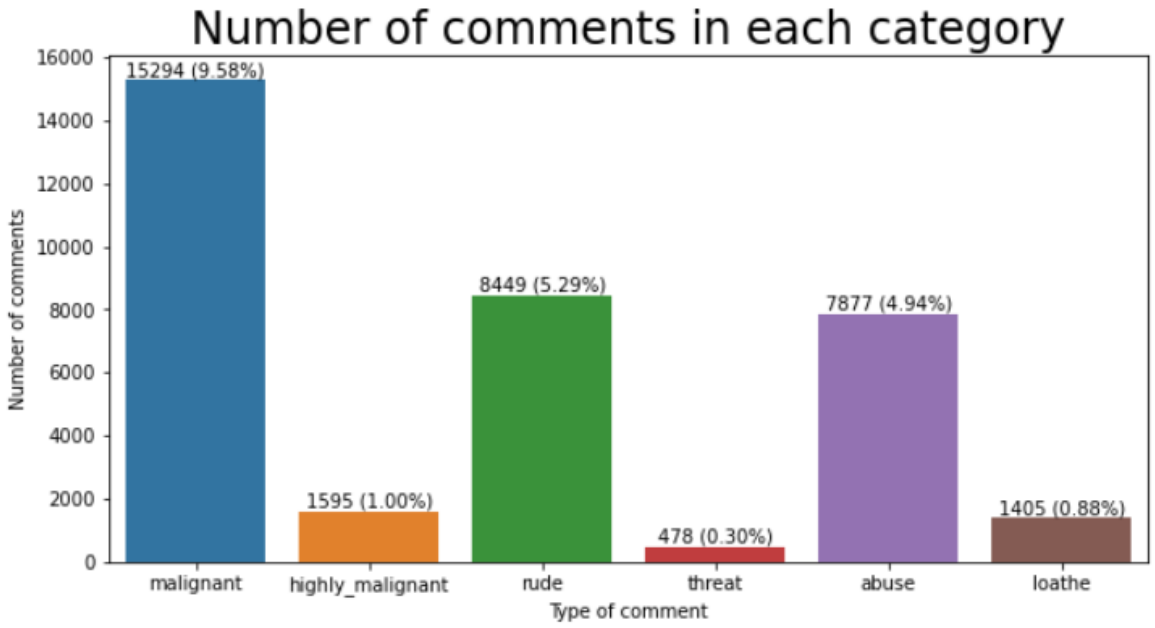
  Social media is an open platform for all users across the globe .The platform which gives the benefit of expressing views of each individual. Analysing each and every person's comments/views/opinions is almost impossible task and hence online platforms faces this disadvantage of hatred/bullying/threat comments on it.

- ## Motivation for the Problem Undertaken

  To understand real world problems where Machine Learning and Data Analysis can be applied to help organizations in various domains from which one domain is Social media.

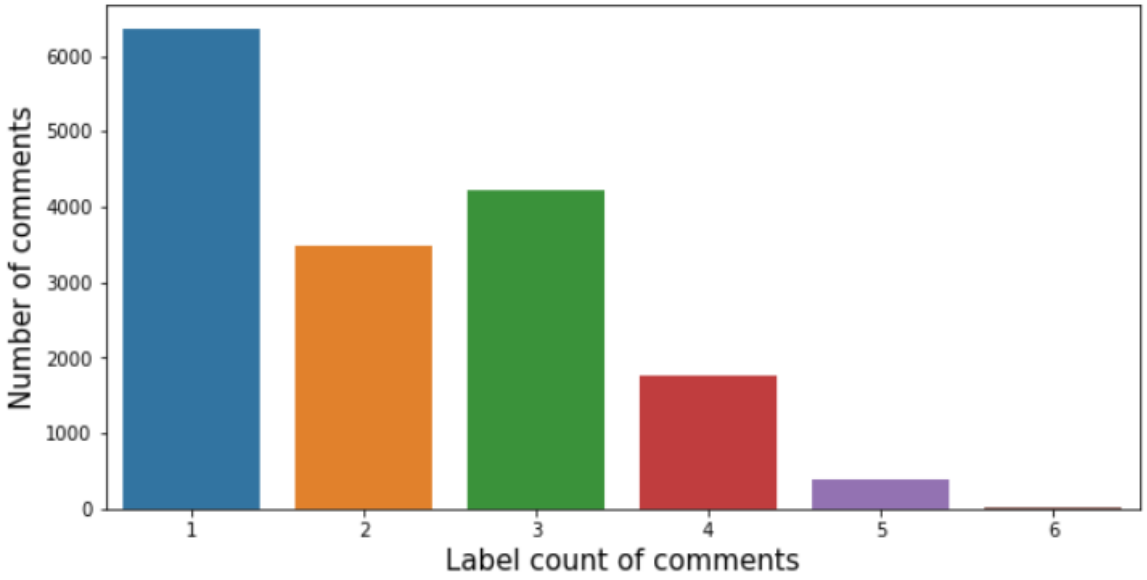
Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem
  - Dataset contained 159571 comments which were raw and uncleaned.
  - Dataset was highly imbalanced with 90% of clean comments and 10 % of unexpected comments distributed in 6 categories.
  - There were no null values.
  - There were 6 target labels :- Highly_maligant , malignant, rude ,threat , abuse , loathe. The dataset contained comments of each of these categories.
  - To better analyse the distribution of comments among all categories I have used bar plot as below showing number and percentage of comments belonging to that label:



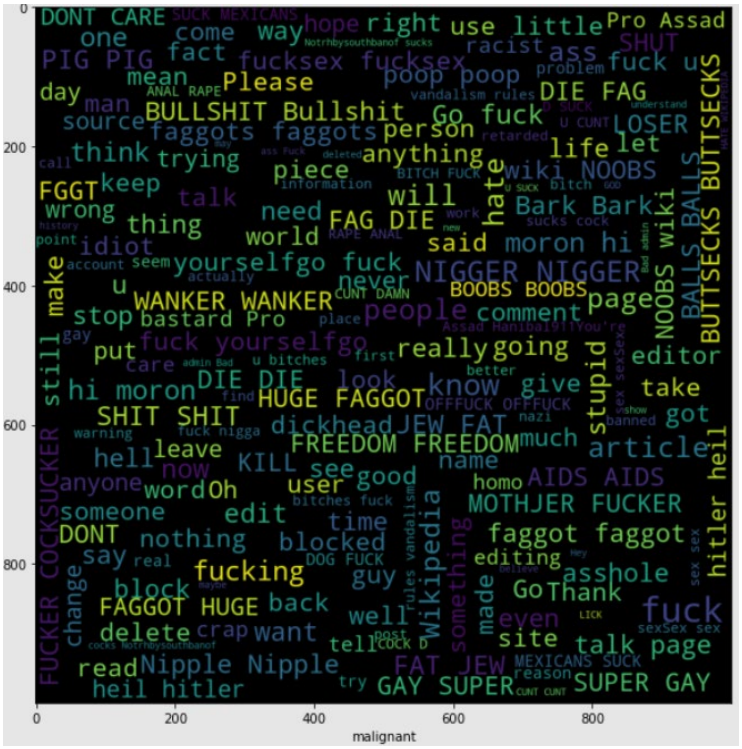
    - Highest unwanted comments were of malignant type followed by rude and abuse.
    - Threat comments were least in number with only 0.33%.
  - Some of the comments belonged to more than one category i.e comments belong to different label (multilabel comments) to check this I have used below plot:

```
0    89.832112
1     3.985687
3     2.637697
2     2.180847
4     1.102957
5     0.241272
6     0.019427
Name: all_label_sum, dtype: float64
```



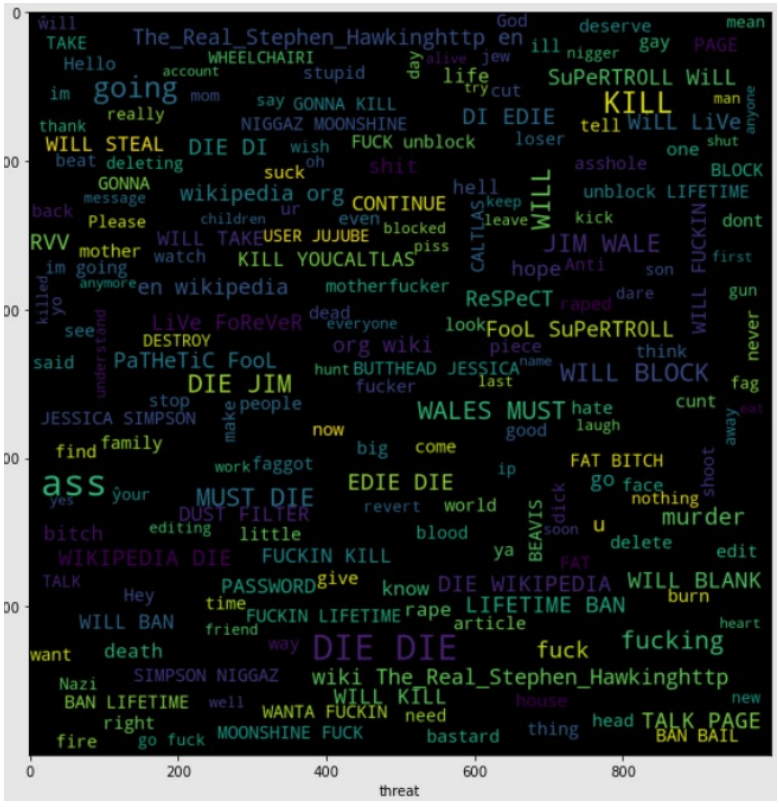
- Highest multilabel comments contained 3 labels.
- Approx 31 comments belong to all the labels.

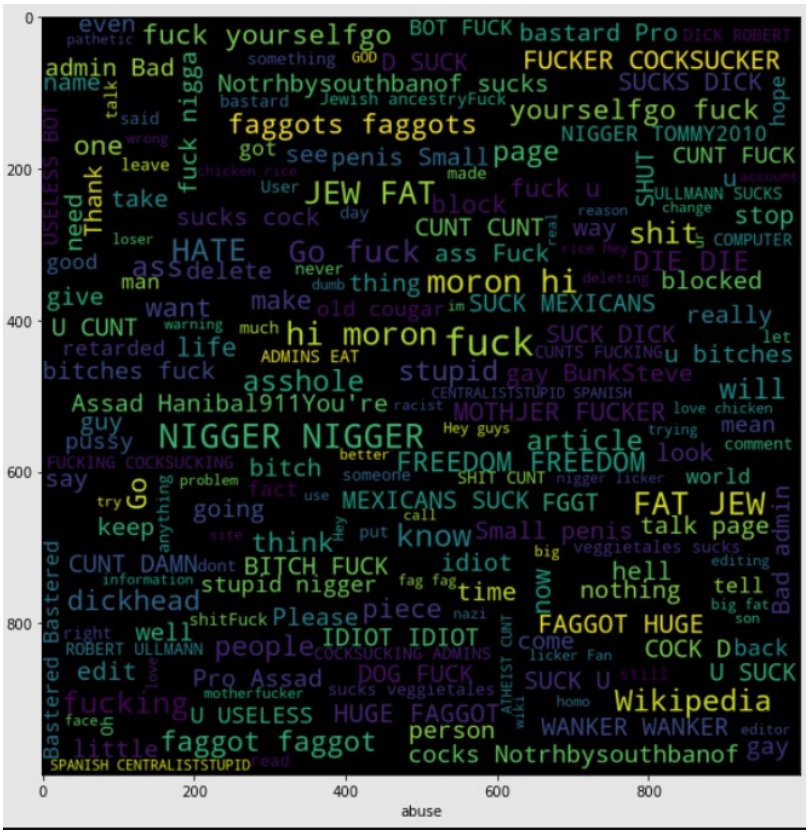
➢ To get an idea of the kind of words present in each label I used wordcloud object for each label which gave below results showing the top 200 words belonging to each label:

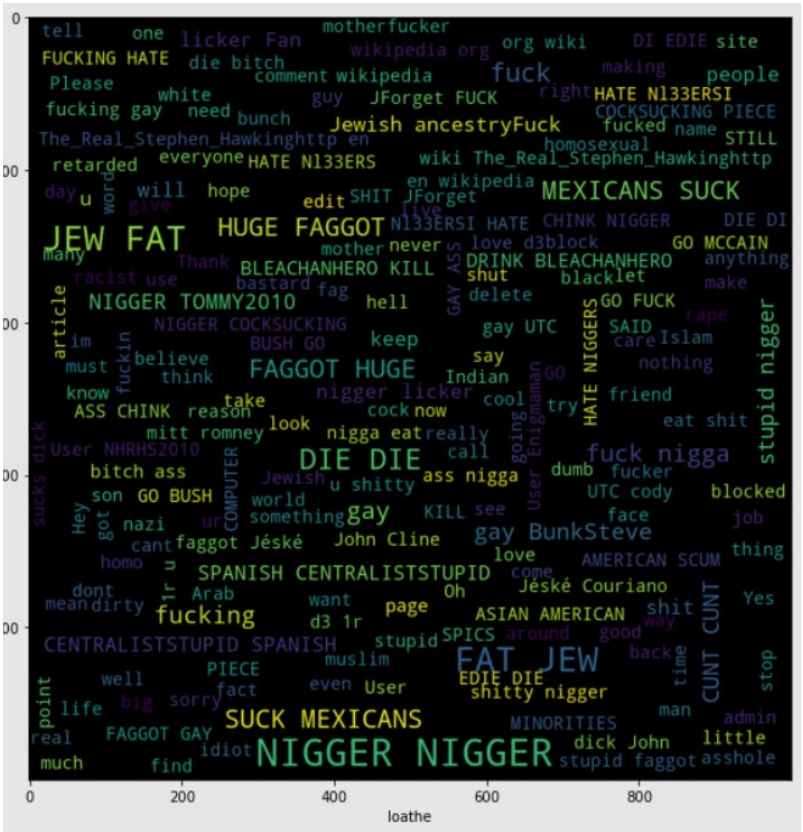

malignant


highly_malignant


rude
0
200
400
600
800
problem
IDIOT IDIOT
information DICK ROBERT
BUTTSECKS BUTTSECKS
MOTHJER FUCKER
blocked
FUCK WIKIPEDIA
U SUCK
CENTRALISTSTUPID SPANISH
FREEDOM FREEDOM
fucking
going
back
give
FAT JEW
people
time
COCKSUCKING ADMINS
shitFuck
fact
KILL
bitches fuck
hope
come
dickhead
SECURITYFUCK HOMELAND
let take
anyone
piece
wikipedia org
something
faggots faggots
say
Thank
MEXICANS SUCK
tell
penis Small
Chester MarcolFuck
OFFFUCK OFFFUCK
DIE DIE
talk page
NIGGER NIGGER
pussy
COCK D
D SUCK
SHIT CUNT
SUCK DICK
Bastered Bastered
well stop
need
fuck yourselfgo
stupid nigger
fuck u
made
leave
U CUNT
bitch
PERSON
U USELESS
ass
fuck nigga
Nipple Nipple
RAPE ANAL
Go fuck
way
CUNT FUCK
SHUT
ULLMANN SUCKS
article
page
one
FUCKER COCKSUCKER
cocks Notrhbysouthbanof
good
put
anything
think
HUGE FAGGOT
edit
guy
stupid
poop poop
talk
COMPUTER
hell
Assad Hanibal911You're
life
CUNT DAMN
SUCKS DICK
asshole
u bitches
block
nothing
wikipedia
keep Pro Assad
JEW FAT
veggietales sucks
gay
even
delete
Notrhbysouthbanof sucks
SUCK MEXICANS
SHIT SHIT
BOOBS BOOBS
know
MarcolFuck Chester
SUCK U
en wikipedia
fucksex fucksex
Small penis
fuck
BITCH FUCK
source
read
right
make
sucks veggietales
said
will
FAGGOT HUGE
thing
FACK
hate
ass Fuck
SPANISH CENTRALISTSTUPID
littleHOMELAND SECURITYFUCK


threat
0
200
400
600
800
Will
TAKE
The_Real_Stephen_Hawkinghttp
en
God
deserve
mean
Hello
WHEELCHAIRI
ill
nigger
gay
PAGE
account
stupid
life
SuPeRTROLL WiLL
going
mom
say GONNA KILL
KILL
really
NIGGAZ MOONSHINE
DI EDIE
tell WILL LiVe
thank
WILL STEAL
DIE DI
wish
FUCK unblock
loser
beat
deleting
suck
shit
asshole
BLOCK
GONNA
wikipedia org
CONTINUE
hell
WILL
unblock LIFETIME
message
back
Please
children
even
blocked
keep
leave
kick
dont
RVV
WILL TAKE
USER JUJUBE
JIM WALE
mother
watch
KILL YOUCALTLAS
piss
hope
Anti
first
im going
anymore
en wikipedia
motherfucker
ReSPeCT
dare
gun
dead
raped
see
LiVe FoReVeR
everyone
look
FooL SuPeRTROLL
never
DESTROY
org wiki
piece
think
said
PaTHeTiC FooL
hunt
BUTTHEAD JESSICA
WILL BLOCK
fag
DIE JIM
fucker
last
stop
people
WALES MUST
hate
cunt
JESSICA SIMPSON
now
good
laugh
find
family
big
come
FAT BITCH
work
faggot
ip
ass
EDIE DIE
go
face
MUST DIE
revert
world
nothing
murder
DUST FILTER
bitch
editing
little
blood
u
delete
WIKIPEDIA DIE
FUCKIN KILL
ya
edit
TALK
PASSWORD
give
know
DIE WIKIPEDIA
WILL BLANK
Hey
time
FUCKIN LIFETIME
rape
LIFETIME BAN
burn
WILL BAN
friend
article
want
death
DIE DIE
fuck
fucking
Nazi
SIMPSON NIGGAZ
wiki The_Real_Stephen_Hawkinghttp
BAN LIFETIME
WILL KILL
WANTA FUCKIN
need
head
TALK PAGE
right
MOONSHINE FUCK
bastard
BAN BAIL
fire
go fuck





- Data Sources and their formats

I got the data from FlipRobo technologies .The dataset contained separate train and test files. There were 159571 train samples and about 153000 samples. There were also a separate file containing description about the features.

- # Data Preprocessing Done

Step 1: Since the 'Id' column was not required , I dropped it and made a separate dataset.

Step 2: Considering only limited samples: Since dataset had 90% comments which were normal , I decided to consider only 50 % of the normal comments considering that all words will be covered in those 50 % comments. This was done since dataset was huge and could have suffered time complexity when fed to a machine learning model.

Step 3: Adding new feature: To better analyse the comments cleaning I added a column showing length of each comment.

Step 4: Converting all text to small case: All comments were converted to small case so that same words with different cases can be treated similarly.

Step 5: Removing Stopwords: Words which do not convey important meaning were removed using stopwords in NLTK library. Along with the default words I also added several unwated characters to the stopwords so that they can also be removed from comments. Below is the code:

```
unwanted_characters =  [")","(", ",","'",".",":",";","``","...","'","!","=","+","-","==","===","====","*","^",'$',"%",
                        "@","{","}"]
Stopwords= set(stopwords.words('English')+ unwanted_characters)
no_stopwords=[]
for sentence in df['comment_text']:
    no_stopwords.append(' '.join([word for word in word_tokenize(sentence) if word not in Stopwords]))
df['comment_text']=no_stopwords
```

Step 6: considering only alphabets and performing lemmatization: To ensure the comments contains only alphabets I performed python Regex operation to skip all words which do not contain alphabets . And finally I performed lemmatization of the word.

Code is as below:

```
from nltk.stem import WordNetLemmatizer
mod_sent=[]
lemma=WordNetLemmatizer()
for sentence in df['comment_text']:
    mod_sent.append(' '.join([lemma.lemmatize(re.sub("[^a-zA-Z]",'',word)) for word in sentence.split()]))
df['comment_text']=mod_sent
```

Step 7: Removing single letter word: Words containing single letters like 'i' are not so important and removed using regex operation as below:

```
c=[]
for i in df['comment_text']:

    c.append(re.sub("\s[a-z]\s"," ",i))
df['comment_text']=c
```

Step 8: Adding a feature: After doing above cleaning of data I added a new feature showing the length of each preprocessed comment. Then I compared the total length of uncleaned data and cleaned data and found the length was reduced to 50 % which means 50% text were unwanted for model.

Step 9: Converting all words to vectors: Since the machine learning model accepts numeric data I then utilized **Word2Vec** word embedding technique to vectorize the words. Detailed steps are as follows:

- First I converted each comment to a list of words present in it and then made a final list containing all those lists. This was done to make the data fit to be fed in a word2vec model as input.

```
sentences= [word_tokenize(sentence) for sentence in df['comment_text']]
```

- In my analysis I have decided to represent each word in 1000 dimensions therefore a word2vec model was prepared as follows.:

```
w2v=Word2Vec(sentences,vector_size=1000,window=5,min_count=1)
```

- Now the words which were not in wor2vec vocabulary were removed from the list of words and while doing so it may be possible that a list gets empty , so empty list were also removed. While doing so I have also stored the indexes of the list which gets empty because we have to remove the same indexes from target label as well to keep the shape of train and test data as same: Below is the code for this task:

```
sentence=[]
index=[]
ind=0
for words in sentences:
    sent=([word for word in words if word in w2v.wv.key_to_index])
    ## checking if list is not empty
    if(len(sent)>0):
        sentence.append(sent)
    else:
        index.append(ind)
    ind+=1
sentences=sentence
```

- I then prepared a method which will take list of words of a sentence as input and give the vectorized form of it. Since there can be multiple words and each word will be converted to 1000 dimensions I decided to take the mean

of all the words of a sentence and return the final vector for each sentence. Below is the 2 line code for that:

```
def sentence_to_vector(sentence,dim):
    return np.mean(w2v.wv[sentence],axis=0).reshape(1,dim)
```

- Finally I got my train vector of dimension (sample size , dim) by passing each list containing words of each sentence to above function:

```
train_vector=np.concatenate([sentence_to_vector(sentence,1000) for sentence in sentences])
```

Step 10 : Scaling the train data: Since its always good to keep the data on the same scale I performed MinMax scaler on my train_vector.

```
from sklearn.preprocessing import MinMaxScaler
minmax=MinMaxScaler()
minmax.fit(train_vector)
train_vector=minmax.transform(train_vector)
```

- ## Data Inputs- Logic- Output Relationships

The input data was raw and object type this data was transformed to the format fit for a machine learning model , using various steps of preprocessing described above. The logic is simply converting all important words of the input to a vector form of fixed dimension and then using it to train the model.

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

In my analysis I have assumed that the target labels are independent of each other and also the data which I have considered (after dropping 50% of normal comments) contains all the words of the whole dataset.

- ## Hardware and Software Requirements and Tools Used

- **Hardware used**: I prepared this complete project on 8 Gb RAM machine to ensure good processing of algorithms since we have huge data. Nothing else was required related to hardware stuff.
- **Software used**: Anaconda installation was the must requirement since it installs two main components needed to develop any datascience project i.e Python programming and Jupyter notebook (a tool used to write python codes ,typically used by all datascientists)
- **Libraries used**: Below are the python library used in this project.
- **Pandas**: To read the Data file in form of data frame and do various data manipulations using pandas Data frame functions.

- **Matplotlib**: This library is typically used to plot the figures for better visualisation of data.
- **Seaborn**: A advanced version of Matplotlib to do data visualisation.
- **Scikit Learn** : This is the most important library for Machine Learning since it contains various Machine Learning Algorithms which are used in this project. Scikit Learn also contains Preprocessing library which is used in data preprocessing. Apart from this it contains very useful joblib library for serialization purpose using which final model have been saved in this project.
- **NLTK:** An important library for word processing ,performing lemmatization etc.
- **Keras :** A wrapper above tensorflow . This library is used to develop deep learning models using ANN ,CNN and RNN.

Model/s Development and Evaluation

- ## Identification of possible problem-solving approaches (methods)

  Since it was clear that it's a binary classification problem containing 6 target labels. Therefore I decided to make six models, each predicting one target. To choose the best model I decided to run my preprocessed dataset on 4 to 5 classification algorithms and also try ANN in deep learning if that can give better result.

- ## Testing of Identified Approaches (Algorithms)

  Below were all the algorithms used for this problem:
  Step 1- LogisticRegression , KNeighborsClassifier ,DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier
  Step 2- ANN model using keras tuners

- ## Run and Evaluate selected models

  In my approach I have first prepared a method which gives all necessary classification metrics of an algorithm like classification metrics , auc_roc score and confusion matrix. This method takes 5 parameters i.e the train_x, train_y, test _x, test_y and the model. Below is the snapshot of this method:

```
def model_performance(train_x,train_y,test_x,test_y,model):
    model.fit(train_x,train_y)
    pred=model.predict(test_x)
    print('Confusion matrix :\n',confusion_matrix(test_y,pred,labels=[1,0]))
    print('\nclassification_report :\n',classification_report(test_y,pred))
    false_positive_rate,true_positive_rate,thresholds=roc_curve(test_y,pred)
    roc_auc=auc( false_positive_rate,true_positive_rate)
    print('\nroc_auc_score :\n',roc_auc)
    #roc_score.append(roc_auc)
    plt.figure(figsize=(10,40))
    plt.subplot(911)
    plt.title(model)
    print(sb.heatmap(confusion_matrix(test_y,pred),annot=True))
    plt.subplot(912)
    plt.title('roc_auc')
    plt.plot(false_positive_rate,true_positive_rate,label='AUC = %0.2f'% roc_auc)
    plt.plot([0,1],[0,1],'r--')
    plt.legend('lower right')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    print('\n\n')
```

Now it become very easy for me to run all the algorithm I want to test. So I then prepared a loop which contains all algorithms I wanted to test and invoked my method by passing all algorithm one by one. The code is a below:

```
Y=df['malignant'].drop(index)

train_x,test_x,train_y,test_y=train_test_split(train_vector,Y,test_size=0.3,random_state=42,stratify=Y,shuffle=True)
models=[LogisticRegression(),KNeighborsClassifier(),DecisionTreeClassifier(),GradientBoostingClassifier(),
        RandomForestClassifier()]
for i,k in zip(models,['LogisticRegression','KNeighbor','DecisionTree','GBM','RandomForest']):
    print('$$$$$$$$$$$$$$$',k,'$$$$$$$$$$$$$$$')
    model_performance(train_x,train_y,test_x,test_y,i)
```

- # Key Metrics for success in solving problem under consideration

I got results of all metrics for all algorithms used, now was the time to decide the evaluation metrics. Looking at the problem it was clear that we had more data where target was ‘0’ and very less data was ‘1’. The problem is concerned about identifying the malignant comments i.e, '1' so if our model predict high number of "False negatives" i.e predicting a abnormal comment (1) as normal(0), this can be a deviation from our aim and also since there are very high number of ‘0’ in my data there is high possibility of ’1’ getting displayed as ‘0’. So I decided to finalize such a model which gives low False negatives (i.e high Recall) and not disturbing Precision much or if possible gives high Recall along with High Precision. Therefore I chose Recall /F1 score as my evaluation metrics for this problem.

Among all the results we got from all the algorithm used using the for loop above, RandomForest was working best since it was giving lowest number of False positives with the Precision of 81%. Below was the results I got for RandomForest:

```
$$$$$$$$$$$$$ RandomForest $$$$$$$$$$$$$
Confusion matrix :
 [[ 2333  2255]
 [  432 24835]]

classification_report :
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     25267
           1       0.84      0.51      0.63      4588

    accuracy                           0.91     29855
   macro avg       0.88      0.75      0.79     29855
weighted avg       0.91      0.91      0.90     29855


roc_auc_score :
 0.7457015180746696
AxesSubplot(0.125,0.808774;0.62x0.0712264)
```

I then decided to tune the parameters of RandomForest to see if there exist more suitable combination of precision and Recall .

To do hyperparameter tuning I prepared a method which performs grid search for the selected algorithm. This method takes 4 parameters i.e independent variable x, The method looks as follows:

```
def tune_parameters(x,y,model,para_dict):
    gds= GridSearchCV(estimator=model, param_grid=para_dict, cv=2,scoring='f1')
    gds.fit(x,y)
    #print('Best score' , gds.best_score_)
    print('Best Parameter', gds.best_params_)
```

Since I knew the model to be hypertuned i.e RandomForestClassifier() therefore I created another method which will give best parameter for the parameter dictionary passed in it.

```
def get_parameter(x,y,Para_Dict_list):
    for each_dict in Para_Dict_list:
        tune_parameters(x,y,RandomForestClassifier(),each_dict)
```

Now it became easy to tune the parameters of the algorithm by just calling above method over a parameter dictionary. Note: I have used scoring f1 to give best possible combination of precision and Recall.

Below is the code how I invoked this method to perform step wise hyperparameter tuning for RamdomForest :

```
para_dict1={'n_estimators':[100,500,100,2000],'max_features':['auto','sqrt']}
tune_parameters(train_vector, Y,RandomForestClassifier(),para_dict1)
```

```
para_dict2={'max_depth':range(5,30,6)}
tune_parameters(train_vector, Y,RandomForestClassifier(),para_dict2)
```

```
para_dict3={'min_samples_split':[2,20,50]}
tune_parameters(train_vector, Y,RandomForestClassifier(),para_dict3)
```

```
para_dict4={'min_samples_leaf':[1,2,5,10]}
tune_parameters(train_vector, Y,RandomForestClassifier(),para_dict4)
```

Again I checked my model using the tuned parameters and got below result:

```
[[ 2305  2283]
 [  419 24848]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     25267
           1       0.85      0.50      0.63      4588

    accuracy                           0.91     29855
   macro avg       0.88      0.74      0.79     29855
weighted avg       0.91      0.91      0.90     29855
```

This was quite better than the untuned model.
Further I wanted to check If we can get better result using Deep learning ANN network. Therefore I prepared a ANN network using keras tuner to tune for the optimum number of layers as well as the number of neurons in each layer. Below is the code for that:

```
from kerastuner.tuners  import RandomSearch
def model_builder(hp):
    model=Sequential()
    for i in range(hp.Int('unit',2,20)):
        model.add(Dense(units=hp.Int('units'+str(i),min_value=32 , max_value=512,step=32), activation='linear'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',[0.001,0.01])),
                  metrics=[Recall(), Precision(),FalseNegatives() , FalsePositives()])
    return model
```

```
### making tuner object declaring objetive as False negative to be minimum
tuner=RandomSearch(
    model_builder,
    objective=kerastuner.Objective("val_false_negatives", direction="min"),
    max_trials=1,
    executions_per_trial=5,
    directory='project1',
    project_name='Malignant'

)
```

```
tuner.search(x_train,y_train, batch_size=50,epochs= 5,validation_data=(x_test,y_test))
```

```
best_model=tuner.get_best_models(num_models=1)
best_model[0].evaluate(x_test,y_test)
```

As per the results I could see the deep learning model increased my Recall to much extent but was giving a very poor precision. Therefore I decided to opt for the RandomForestClassifier model and chosen it as the final model.
Finally since there was 6 labels therefore I prepared 6 models after tuning parameters for each. Each model was for one particular label. Below is the code for only label as example:

```
para_dict_list=[{'n_estimators':[2000],'max_features':['auto','sqrt']},
                {'max_depth':range(5,30,6)},
                {'min_samples_split':[2,20,50]},
                {'min_samples_leaf':[1,2,5,10]}]
## getting best model parameters for 'highly_malignant' label
y_highly_malignant=df['highly_malignant'].drop(index)
get_parameter(train_vector,y_highly_malignant,para_dict_list)
```

```
Best Parameter {'max_features': 'sqrt'}
Best Parameter {'max_depth': 29}
Best Parameter {'min_samples_split': 2}
Best Parameter {'min_samples_leaf': 1}
```

```
# train test split
x_train,x_test, y_train, y_test=train_test_split(train_vector, y_highly_malignant,test_size=0.3,shuffle=True,
                                                 stratify=y_highly_malignant,random_state=1)
print(x_train.shape, y_train.shape, x_test.shape,y_test.shape)
model_highly_malignant=get_results_for_model(x_train, y_train, x_test, y_test, RandomForestClassifier(n_estimators=2000
                                             , max_features='sqrt',max_depth=29,min_samples_split=2, min_samples_leaf=1))
```

```
(69660, 1000) (69660,) (29855, 1000) (29855,)
[[   60   419]
 [   41 29335]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     29376
           1       0.59      0.13      0.21       479

    accuracy                           0.98     29855
   macro avg       0.79      0.56      0.60     29855
weighted avg       0.98      0.98      0.98     29855
```

Similarly it was done for all other labels and 6 models was prepared.
Finally the test data was imported and pre-processed the same way as training data. The cleaned test data was then passed trough the 6 models to predict the 6 categories of the comment. All predictions were then merged into a dataframe to get the output predictions file.

```
predictions_1=model.predict(test_vector)
predictions_2=model_highly_malignant.predict(test_vector)
predictions_3=model_rude_.predict(test_vector)
predictions_4=model_threat.predict(test_vector)
predictions_5=model_abuse.predict(test_vector)
predictions_6=model_loathe.predict(test_vector)
```

```
predictions=pd.DataFrame({'Malignant':predictions_1,'Highly_malignant': predictions_2,
                          'rude':predictions_3,'threat' :predictions_4,
                          'abuse': predictions_5,'loathe': predictions_6})
```

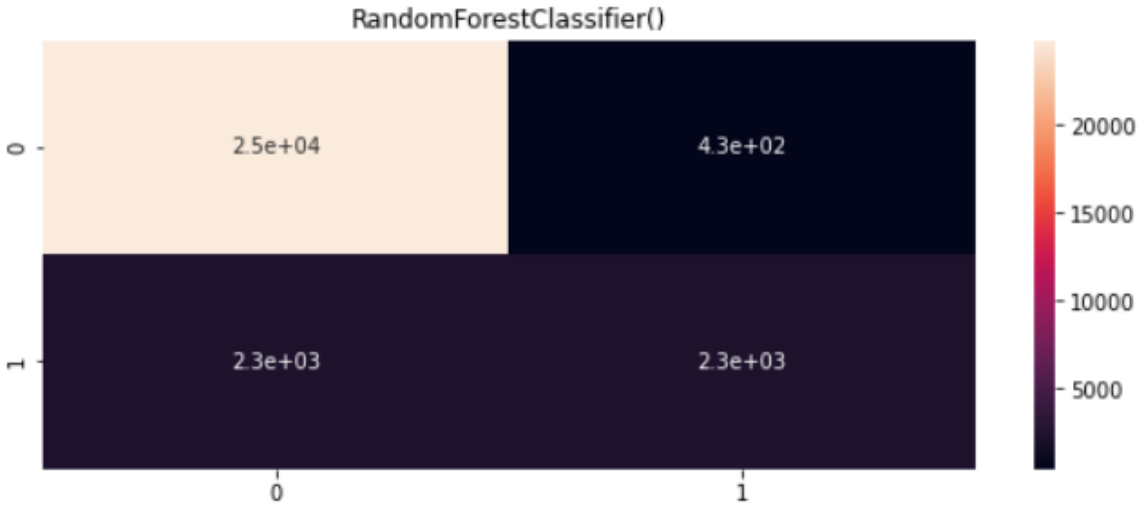
```
predictions.to_csv('Predictions.csv')
```

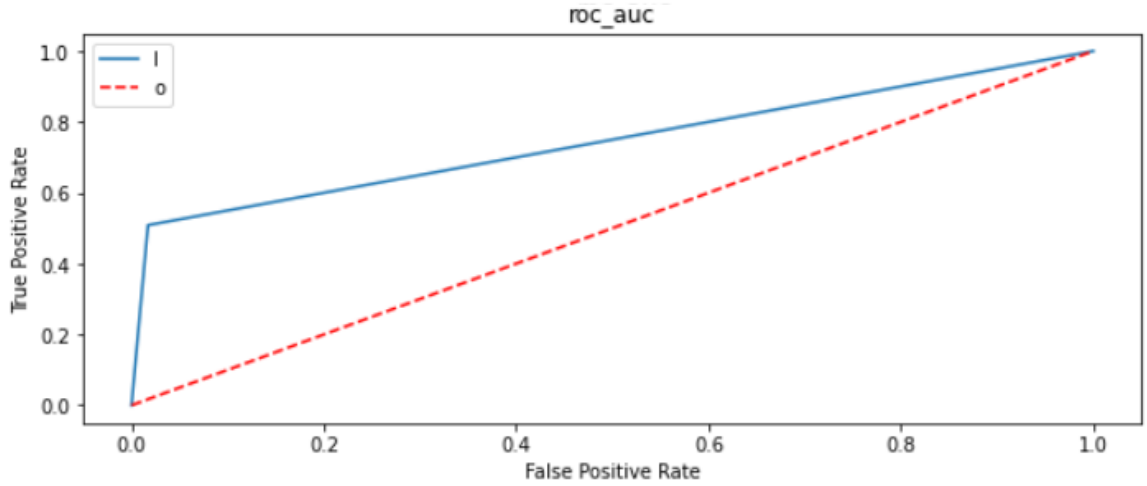
```
predicitions_prob=model.predict_proba(test_vector)
```

```
pd.DataFrame(data=predicitions_prob).to_csv('predicitions_prob.csv')
```

- Visualizations

  As already seen, in the method I have used seaborn library for plotting the confusion matrix and ROC-AUC curve for better visualisation of the results obtained from them. Below is the plot of confusion matrix for finally chosen RandomForest algorithm:





- Interpretation of the Results
  - The confusion matrix gives the number of TP (2300) ,FP(2300) ,FN(430) and TN (25000).
  - It implies total 50000 rows were considered for testing out of which 2730(2300+ 430) were incorrect and 27300 were correctly predicted by our model.
  - The ROC-AUC curve shows the ability to distinguish between 0 and 1 labels. In my final model I got ROC-AUC score of 75% which implies that probability of my model to distinguish between 0 and 1 is 0.75

CONCLUSION

- ## Key Findings and Conclusions of the Study

  Dataset was highly imbalanced with very high number normal comments (90%) comparison to abnormal comments.
  Not all comments had only one target label. Some words were present in all type of comments.

- ## Learning Outcomes of the Study in respect of Data Science

  I found visualisation a very useful technique to infer insights from dataset especially in case of large datasets.
  Use of Word2Vec model was very helpful in text to numeric conversion.

  In case of large datasets (like this one) it takes lot of time to tune the parameters of selected algorithm using grid search especially for algorithms like gradient Boosting and all ensemble techniques if we pass all the parameters at one go. Therefore I decided to tune one parameter at a time using grid search and I think comparatively it took me less time.

Limitations of this work and Scope for Future Work

Since the dataset is large the time taken to train the models was more.

Since the dataset had very less abnormal comments the trained model will be limited in scope for abnormal comments. More data of abnormal comments can definitely help improving the results of the model.