MALIGNANT COMMENTS CLASSIFICATION

Submitted by:

Aaron D'souza

ACKNOWLEDGMENT

References:

Paper refereed:
https://www.researchgate.net/publication/349929587_Machine_learning_methods_for_toxic_comment_classification_a_systematic_review/link/6048146592851c077f2b0b29/download

Link (Handling Imbalanced Datasets SMOTE Technique):
https://youtu.be/dkXB8HH_4-k

Link (github reference): https://github.com/anmolchawla/Kaggle-Toxic-Comment-Classification-Challenge/blob/master/Data%20Preprocessing%20and%20Exploratory%20Data%20Analysis.ipynb

Link (Paper):
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6909170.pdf

INTRODUCTION

- ## Business Problem Framing

Nowadays users leave numerous comments on different social networks, news portals, and forums. Some of the comments are toxic or abusive.

Due to numbers of comments, it is unfeasible to manually moderate them, so most of the systems use some kind of automatic discovery of toxicity using machine learning models.

Toxic comments are defined as comments that are rude, disrespectful, or that tend to force users to leave the discussion.

If these toxic comment can be automatically identified, we could have safer discussions on various social networks, news portals, or online forums.

Manual moderation of comments is costly, in effective, and sometimes infeasible. Automatic or semi-automatic detection of toxic comment is done by using different machine learning methods.

- ## Conceptual Background of the Domain Problem

  The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users.

  Although researchers have found that hate is a problem across multiple platforms, there is a lack of models for online hate detection.

  Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

  Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

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

- ## Review of Literature

  Due the penetration of the internet in all domains of life which has led to increase of people's participation actively and give remarks as an issue of communicating their concern/feedback/opinion in various online forums.

  Although most of the times these comments are helpful for the creator to extemporize the substance that is being provided to people, but sometimes these may be abusive and create hatred-feeling among the people.

  Thus as these are openly available to the public which is being viewed from various sections of the society, people in different age groups, different communities and different socio-economic background, it becomes the prime responsibility of the content-creator (the host) to filter out these comments in order to stop the spread of negativity or hatred within people.

  Lately there has been many cases in which the growing menace of hate and negativity has been witnessed in the online platforms especially social media as such, many governments around the world has seen the rise of cases related to cyber bullying that has led to spread of hatred and violence.

- ## Motivation for the Problem Undertaken

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms.

Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

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

Analytical Problem Framing

- ## Mathematical/ Analytical Modelling of the Problem

The goal is to build a model that will extract information from text-based data and predict whether the comment was toxic or non-toxic.

Models used for extracting information from text based reviews:

- Bags of words model
- Term Frequency-Inverse Document Frequency model (TF-IDF)

Classification model used:

- Naive Bayes
- Logistic Regression.
- Random Forest.
- Linear SVC.

- ## Data Sources and their formats

  The data was scrapped using selenium web-driver and than saved in

  .CSV (comma-separated values) format.

  A total of two data-types in the dataset Int and Object.

  The data set has a total of 8 columns:

  Malignant: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.

  Highly Malignant: It denotes comments that are highly malignant and hurtful.

  Rude: It denotes comments that are very rude and offensive.

  Threat: It contains indication of the comments that are giving any threat to someone.

  Abuse: It is for comments that are abusive in nature.

  Loathe: It describes the comments which are hateful and loathing in nature.

  ID: It includes unique Ids associated with each comment text given.

  Comment text: This column contains the comments extracted from various social media platforms.

```
1 df.head()
```

	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

Fig.1 Data-frame Sample

- Data Preprocessing Done

Step 1 -->

Checking for null values in the dataset

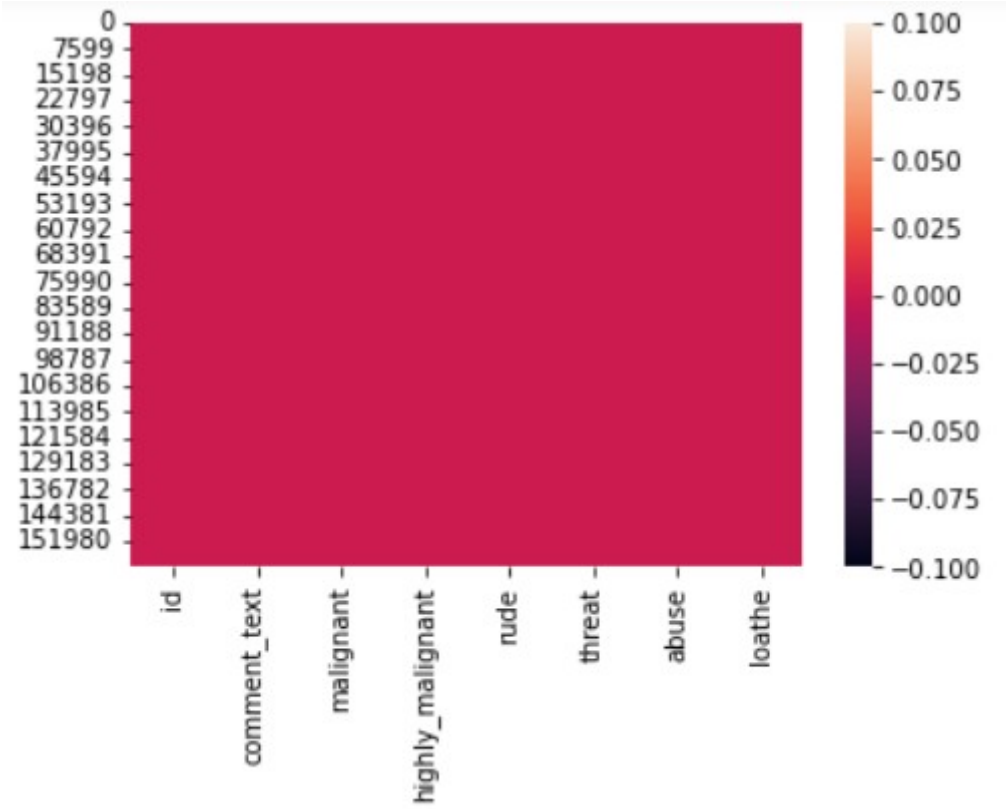


Fig.2 No Null Values.

Step 2 -->

Creating a single column which will have two values Toxic or Non-Toxic

```
cols=['malignant', 'highly_malignant', 'rude', 'threat','abuse', 'loathe']
```

```
# Lets create a single column which will have two values Toxic or non toxic

df['Toxic']=df[cols].max(axis=1).apply(lambda x: "Toxic" if x == 1 else "Non-Toxic")
```

Fig.3 Creating a new column.

Step 3 -->

The Dataset is highly imbalanced.

```
# Checking if the data is balanced or not

clean = len(df[df.Toxic=="Non-Toxic"])
toxic = len(df[df.Toxic=="Toxic"])
```

```
print("Toxic Comments count:",toxic)
print("toxic percentage %.2f"%(toxic/len(df)*100))
```

```
Toxic Comments count: 16225
toxic percentage 10.17
```

Fig.4 Imbalanced Dataset.

Step 4 -->

Text cleaning functions that replace short forms to full forms

```
def decontracted(phrase):
# specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can't", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Fig.5 Remove Short-forms.

Step 5 -->

To remove the <.?/; and any numerical values from the text and applying the function.

```
def stripunc(X):
    return re.sub('[^A-Za-z]+',' ', str(X), flags=re.MULTILINE|re.DOTALL)
```

```
comm = comm.apply(decontracted) #cleaning functions are being applied on the text

comm = comm.apply(stripunc)
```

Fig.6 Removing symbols.

Step 6 -->

Removing Stop-words from the user comments.

```
#spacy stopwords....
stop_rots = sp.Defaults.stop_words
```

```
#removing some words because it may change the non-toxic to toxic....
stop_rots.remove("n't")

stop_rots.remove("no")

stop_rots.remove("not")
```

Fig.7 Removing stop-words.

Step 7 -->

Applying Lemmatizer on user comments.

```
def lemmatize(text):
    lm= WordNetLemmatizer()
    text = text.apply(lambda x: " ".join([lm.lemmatize(word,pos='v') for word in x.split(" ") if
    return text
```

```
df['comments'] = lemmatize(comm)
```

Fig.8 Applying Lemmatization.

Step 8 -->

Train Test Splitting of data.

```
# Train Test Split

from sklearn.model_selection import train_test_split

X_train, X_test,y_train,y_test = train_test_split(X, y, test_size=0.20,random_state =0, shuffle =F
```

Fig.9 Train Test Split.

Step 9 -->

Applying TF-IDF vectoriser. TF-IDF vectoriser has been used as it is far better than bag of words approach.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(ngram_range=(1,2),min_df=3, max_df=0.9,use_idf=True,smooth_idf=1, sublinear_t
```

```
train = tf.fit_transform(X_train['comments'])
```

```
test = tf.transform(X_test['comments'])
```

Fig.10 Applying TF-IDF vectoriser.

Step 10 -->

Applying SMOTE for the synthetic data points generation because of the class imbalance problem.

```
# applying the smote to the miority class only.

sm = SMOTE(random_state=10,sampling_strategy='minority',k_neighbors=15,n_jobs=-1)

sam_train, sam_y = sm.fit_resample(train, y.ravel())  #train is tfidf output maxtrix and y is nump
```

Fig.11 Applying SMOTE.

- ## Hardware and Software Requirements and Tools Used

Hardware used:

OS: Windows 10 Home Single Language 64 bit

Ram: 8 GB

Processor: Intel I5

Software used:

Jupyter Notebook

Model/s Development and Evaluation

- ## Testing of Identified Approaches (Algorithms)

Models used for extracting information from text based reviews:

▪ Bags of words model:

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multi set) of its words, disregarding grammar and even word order but keeping multiplicity. The bag of words model has also been used for computer vision.

The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier.

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Fig.12 BOW Example.

▪ Term Frequency-Inverse Document Frequency model (TF-IDF):

In information retrieval, tf–idf, TF*IDF, or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a

weighting factor in searches of information retrieval, text mining, and user modeling.

The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

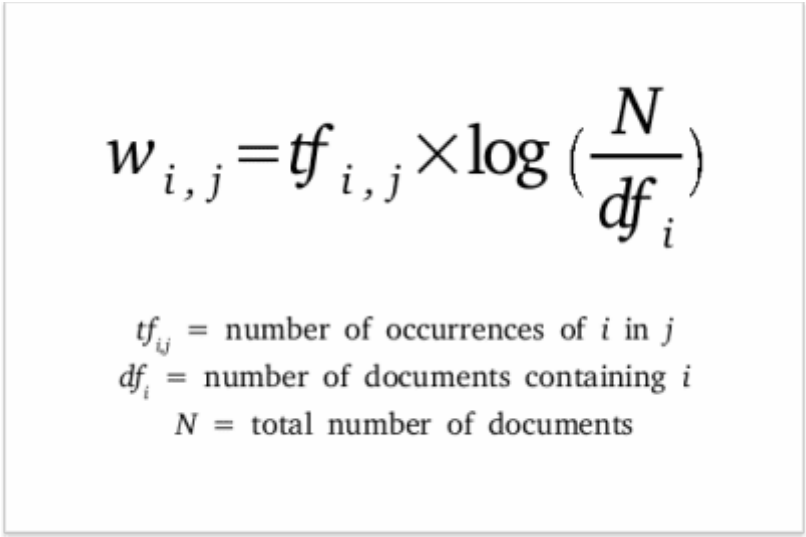


Fig.13 TFIDF Formula.

Classification model used:

▪ Naive Bayes:

A naive Bayes classifier is an algorithm that uses Bayes' theorem to classify objects. Naive Bayes classifiers assume strong, or naive, independence between attributes of data points. Popular uses of naive Bayes classifiers include spam filters, text analysis and medical diagnosis. These classifiers are widely used for machine learning because they are simple to implement.

Logistic Regression:

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Random Forest Classification:

Random forest is a flexible, easy to use algorithm, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

Linear SVC:

SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

- Run and Evaluate selected models

Logistic Regression

Hyper-parametric Tuning

```
# Hyperparameter tunning using Grid Search CV'''

model = LogisticRegression(class_weight='balanced')
penalty = ['l2','l1']
c_values = [10, 1.0, 0.1]
solvers = ['lbfgs', 'sag']

# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           cv=5,verbose=1, scoring='f1_macro',error_score=0)

grid_result = grid_search.fit(sam_train, sam_y)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

Best Parameters, estimators and best score for Logistic regression.

```
print("Best params are:",grid_result.best_params_)
```

```
Best params are: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
print("Best estimators are:",grid_result.best_estimator_)
```

```
Best estimators are: LogisticRegression(C=10, class_weight='balanced')
```

```
#predciting the test set...
best = grid_result.best_estimator_.predict(test)
pre = grid_result.predict(test)
```

```
print("Accuracy on test set is: %.2f"%accuracy_score(pre, y_test))
```

```
Accuracy on test set is: 0.93
```

Classification Report for Logistic Regression:

```
print("Classification Report....")
target_names = ['0','1']
print(classification_report(y_test, pre, target_names=target_names))
```

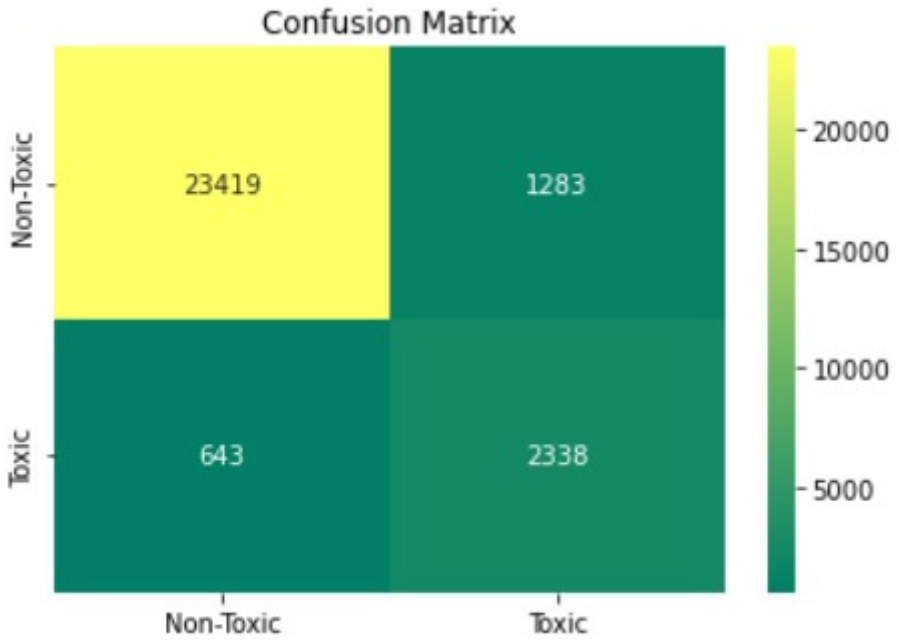
```
Classification Report....
              precision    recall  f1-score   support

           0       0.97      0.95      0.96     24702
           1       0.65      0.78      0.71      2981

    accuracy                           0.93     27683
   macro avg       0.81      0.87      0.83     27683
weighted avg       0.94      0.93      0.93     27683
```

F1- Score with Logistic regression is 0.96 for non-toxic and 0.71 for toxic

Confusion matrix for Logistic Regression:



Random Forest Classification

Hyper-parametric Tuning:

```
# Hyperparameter tunning using Grid Search CV

params ={'min_samples_leaf': [5],'min_samples_split': [12],'n_estimators': [100,150]}

model = GridSearchCV(RandomForestClassifier(oob_score=True,n_jobs=-1), param_grid=params,cv=5, n_j
                     scoring='f1_macro',return_train_score=True,verbose=1)
model = model.fit(sam_train, sam_y)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

Best Parameters, estimators and best score for Random forest classification.

```
print("Best params are:",model.best_params_)
```

```
Best params are: {'min_samples_leaf': 5, 'min_samples_split': 12, 'n_estimators': 150}
```

```
print("Best estimators are:",model.best_estimator_)
```

```
Best estimators are: RandomForestClassifier(min_samples_leaf=5, min_samples_split=12,
                       n_estimators=150, n_jobs=-1, oob_score=True)
```

```
print("Best score is:",model.best_score_)
```

```
Best score is: 0.9188615346040716
```

Classification Report for Random forest classification:

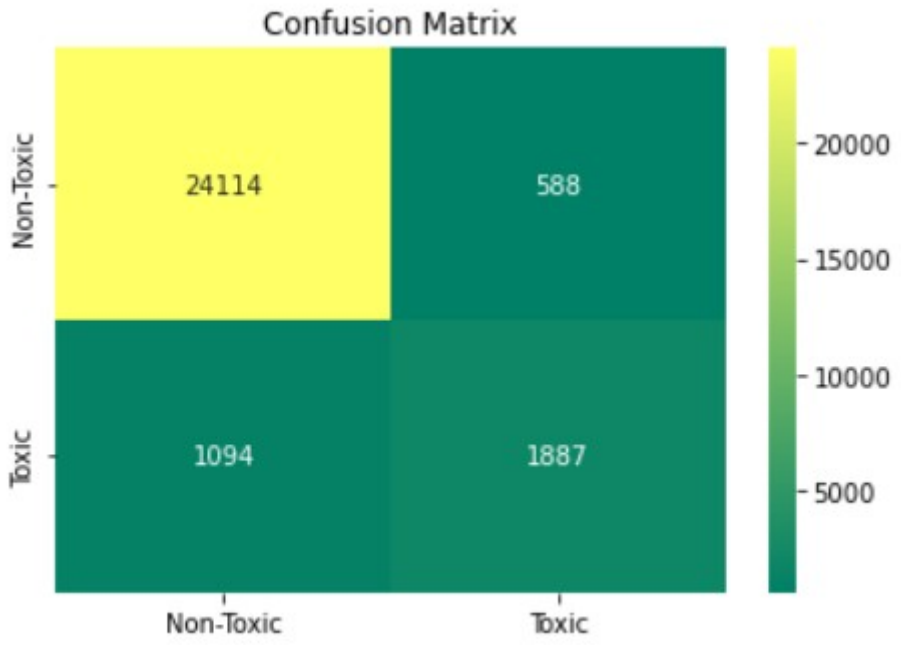
```
print("Classification Report....")
target_names = ['0','1']
print(classification_report(y_test, pred, target_names=target_names))
```

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

           0       0.96      0.98      0.97     24702
           1       0.76      0.63      0.69      2981

    accuracy                           0.94     27683
   macro avg       0.86      0.80      0.83     27683
weighted avg       0.94      0.94      0.94     27683
```

Confusion matrix for Random Forest Classification:



Linear SVC

Hyper-parametric Tuning:

```
from sklearn.svm import LinearSVC

#Hyperparameter tunning using Grid Search CV

params ={'C': [0.01,0.1,1.0,10],'max_iter': [1000,1500],'dual':[True,False]}

clf = GridSearchCV(LinearSVC(class_weight='balanced'), param_grid=params,cv=5, n_jobs=-1,
                   scoring='f1_macro',return_train_score=True,verbose=1)

clf = clf.fit(sam_train, sam_y)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

Best Parameters, estimators and best score for Linear SVC:

```
print("Best params are:",clf.best_params_)
```

```
Best params are: {'C': 10, 'dual': True, 'max_iter': 1000}
```

```
print("Best estimators are:",clf.best_estimator_)
```

```
Best estimators are: LinearSVC(C=10, class_weight='balanced')
```

```
print("Best score is:",clf.best_score_)
```

```
Best score is: 0.9653922645133395
```

Classification Report for Linear SVC:

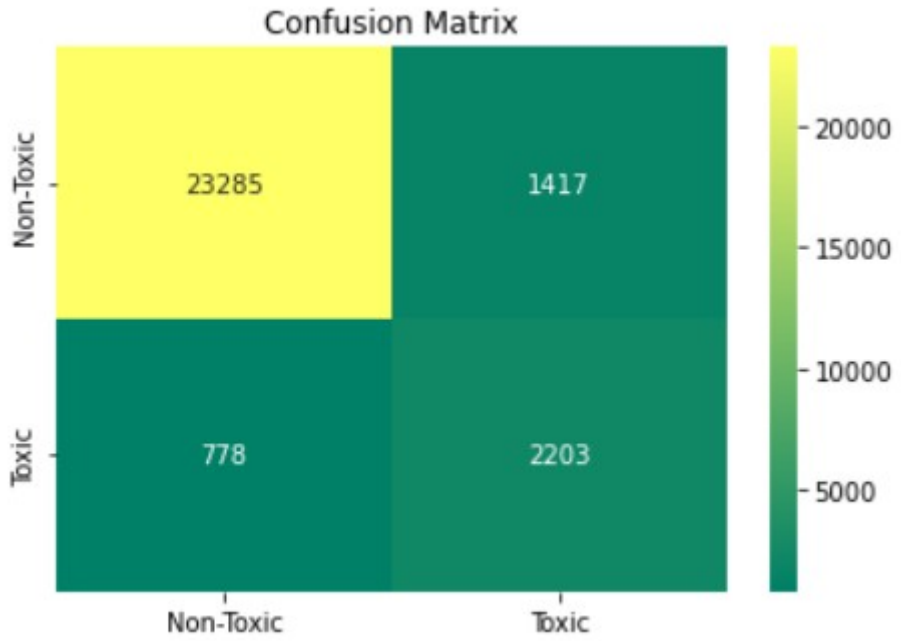
```
print("Classification Report....")
target_names = ['0','1']
print(classification_report(y_test, svc_pred, target_names=target_names))
```

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

           0       0.97      0.94      0.95     24702
           1       0.61      0.74      0.67      2981

    accuracy                           0.92     27683
   macro avg       0.79      0.84      0.81     27683
weighted avg       0.93      0.92      0.92     27683
```

Confusion Matrix for Linear SVC



Naive Bayes

Hyper-parametric Tuning:

```
#Hyperparameter tunning using Grid Search CV

params = {'alpha':[10,100]}
nav = GridSearchCV(MultinomialNB(), param_grid=params,cv=5,n_jobs=-1,
                   scoring='accuracy',verbose=1)
nv = nav.fit(sam_train, sam_y)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

Best Parameters, estimators and best score for Naive Bayes:

```
print("Best params are:",nav.best_params_)
```

```
Best params are: {'alpha': 10}
```

```
print("Best estimators are:",nav.best_estimator_)
```

```
Best estimators are: MultinomialNB(alpha=10)
```

```
print("Best score is:",nav.best_score_)
```

```
Best score is: 0.8777655793860468
```

Classification Report for Naive Bayes:

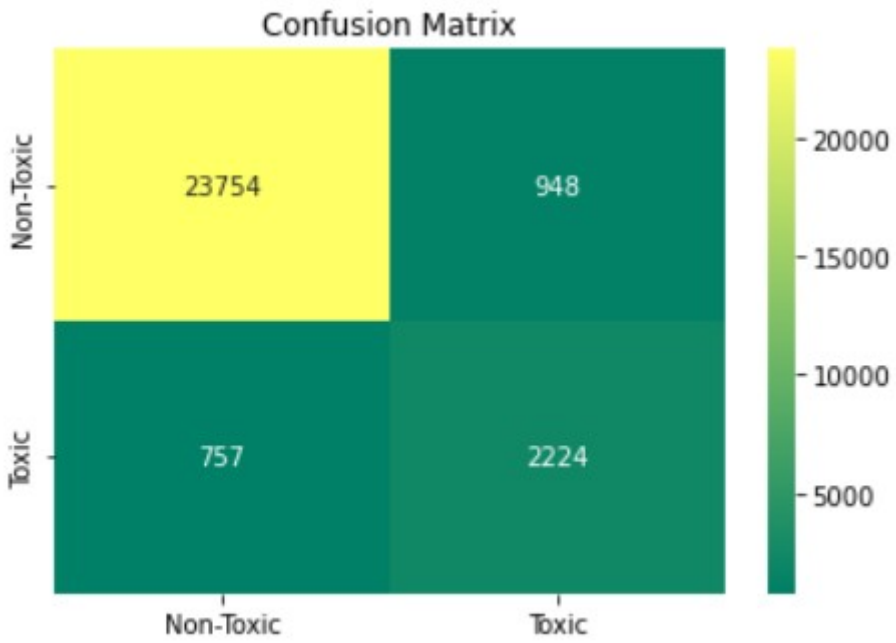
```
print("Classification Report....")
target_names = ['0','1']
print(classification_report(y_test, pr, target_names=target_names))
```

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

           0       0.97      0.96      0.97     24702
           1       0.70      0.75      0.72      2981

    accuracy                           0.94     27683
   macro avg       0.84      0.85      0.84     27683
weighted avg       0.94      0.94      0.94     27683
```

Confusion Matrix for Naive Bayes(Multinomial):



Visualizations

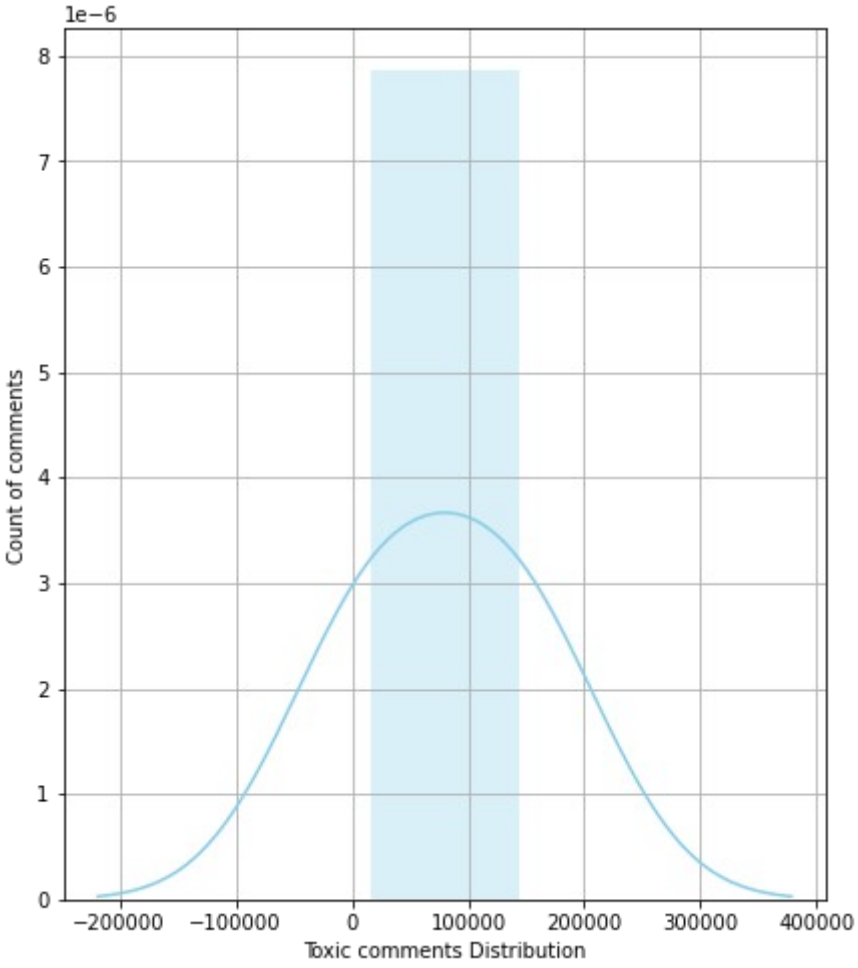


Fig.13 Toxic comment distribution.

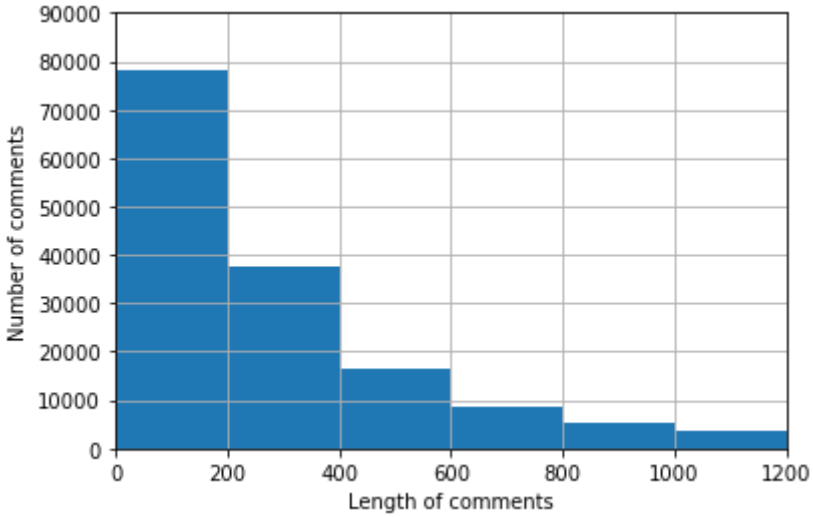


Fig.15 Mean length of comment before cleaning data: 394.139

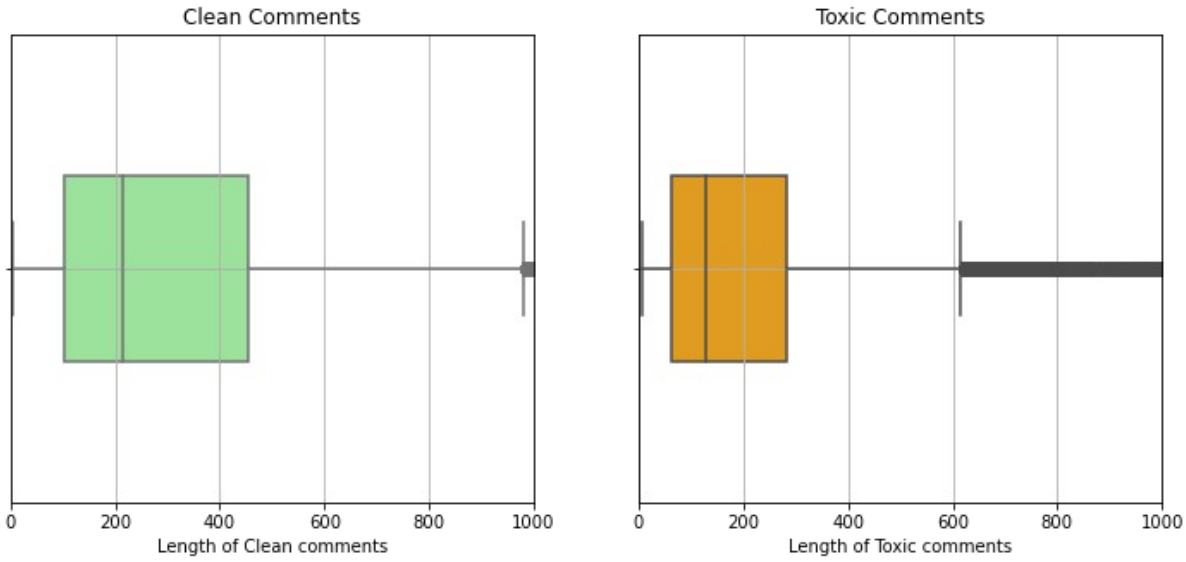


Fig.16 Clean comments have higher word length then toxic comments.

Fig.17 Word-cloud clean comments vs toxic comments.

CONCLUSION

- ## Key Findings and Conclusions of the Study

  This project has discussed four approaches to discuss four machine learning algorithms (classification) and compared various metrics like Accuracy, F1 score, Recall and Precision.

  Four classification algorithms were used Logistic Regression, Naive Bayes (Multinomial), Linear SVC and Random Forest classifier out of this four algorithms Naive Bayes (Multinomial) gave the best results.

- ## Limitations of this work and Scope for Future Work

  In further research we can use complex Deep Learning algorithms to further increase the accuracy.