DATA Trained Education

“Malignant Comment Classification Project”

1. Problem Definition

1.1 Project Overview

Online platforms when used by normal people can only be comfortably used by them only when they feel that they can express themselves freely and without any reluctance. If they come across any kind of a malignant or toxic type of a reply which can also be a threat or an insult or any kind of harassment which makes them uncomfortable, they might defer to use the social media platform in future. Thus, it becomes extremely essential for any organization or community to have an automated system which can efficiently identify and keep a track of all such comments and thus take any respective action for it, such as reporting or blocking the same to prevent any such kind of issues in the future.

In this, we try to recognize the intention of the speaker by building a model that’s capable of detecting different types of toxicity like threats, abuse, insults, and loathe-based hate. Moreover, it is crucial to handle any such kind of nuisance, to make a more user-friendly experience, only after which people can actually enjoy in participating in discussions with regard to online conversation.

1.2 Problem Statement

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 behavior.

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

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

1.3 Data Set Description

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes:

- 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.

This project is more about exploration, feature engineering and classification that can be done on this data. Since the data set is huge and includes many categories of comments, we can do good amount of data exploration and derive some interesting features using the comments text column available.

1.4 Evaluation Metrics

I chose Naive Bayes as our baseline model, specifically Multinomial Naive Bayes.

Also, since i want to compare between different models, especially models that perform well in text classification. Thus, i chose to compare Multinomial Naive Bayes with Logistic Regression and Linear Support Vector Machine

Our main metric for measuring model performance is F1-score, since we have 6 labels, the F1-score would be the average of 6 labels. We will also take other metrics into consideration while evaluating models, e.g, Hamming loss and recall.

2. Analysis

2.1 Data Exploration

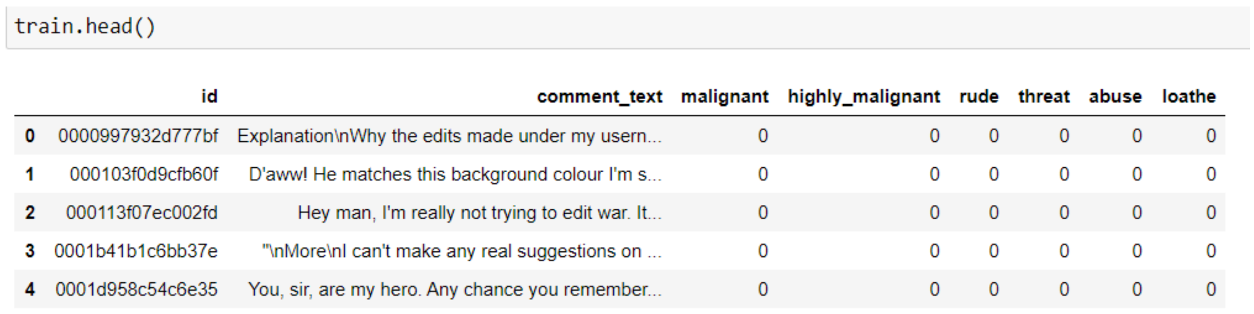
The data set includes:

- 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.

Out of these fields, the comment text field will be preprocessed and fitted into different classifiers to predict whether it belongs to one or more of the labels/outcome variables (i.e., malignant, highly_malignant, rude, threat, abuse and loathe.

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

We have a total of 151203 samples of comments and labelled data, which can be loaded from train.csv file. The first 5 samples are as follows:



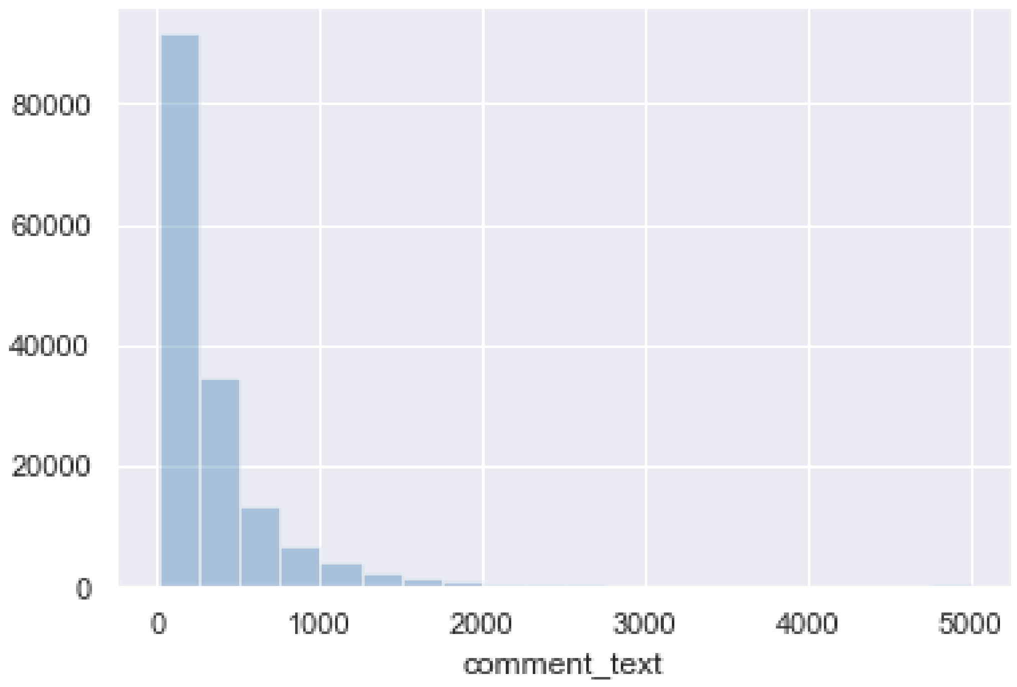
```
train.head()
```

	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

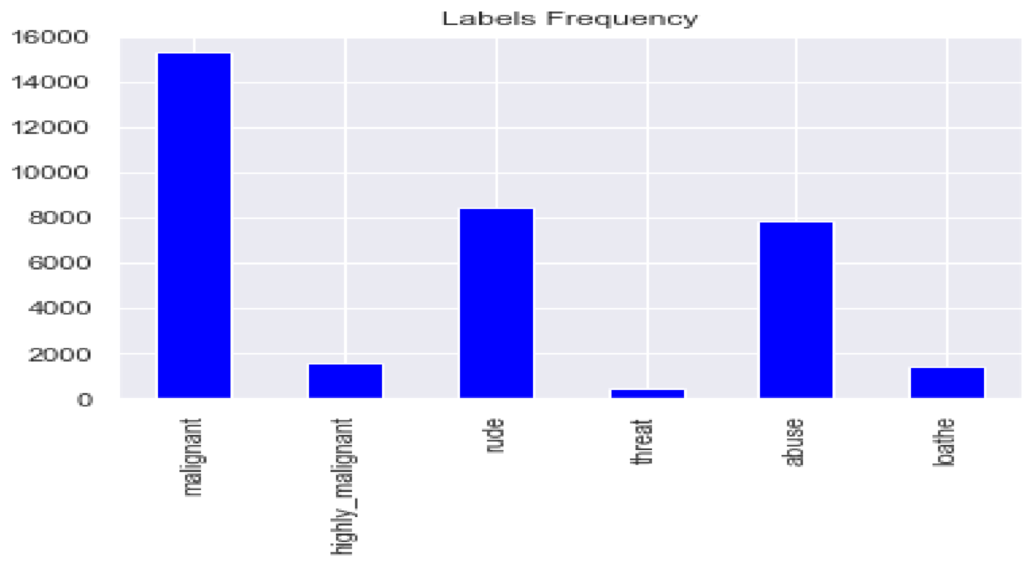
one more very crucial aspect of the dataset is noticing the frequency occurrence of multi labelled data. We can easily notice that approximately every 1 data out of 10 is malignant (9000 samples), every 1 in 20 samples is rude and abuse (5000 samples), but the occurrences of sample being highly_malignant, threat and loathe is extremely rare (800-900 out of 90000 samples). Overall, 9790 samples are those which have at least one label and 5957 samples have 2 or more labels.

2.2 Exploratory Visualization

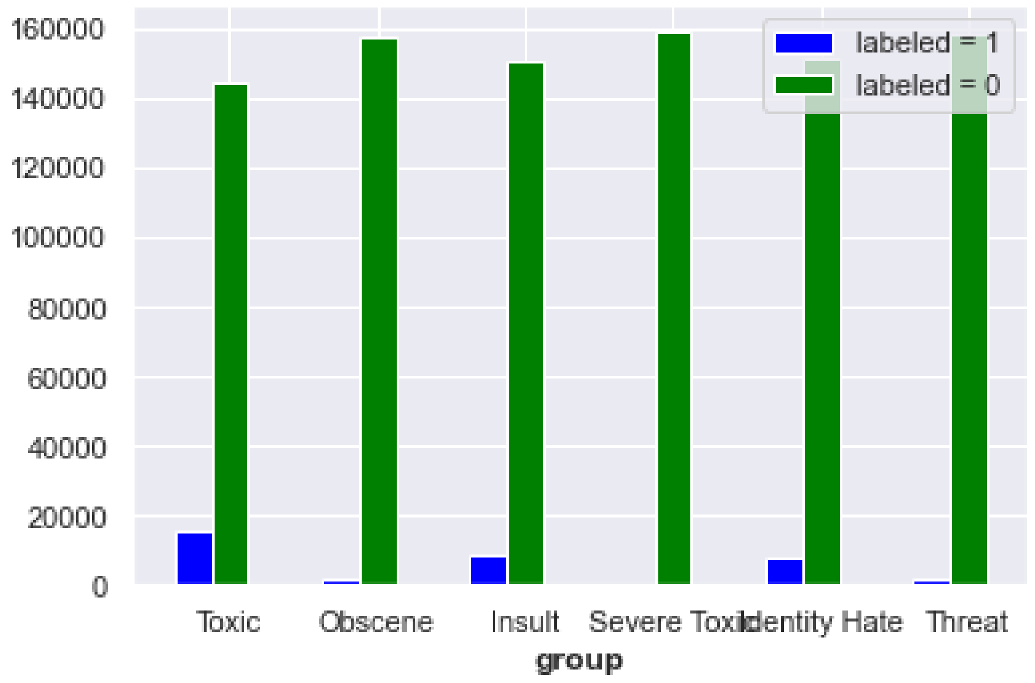
Below is a plot showing the comment length frequency. As noticed, most of the comments are short with only a few comments longer than 1000 words.



Exploratory shows that label Malignant has the most observations in the training dataset while threat has the least.



Below is the plot for the labeled data frequency. There is significant class imbalance since majority of the comments are considered non-toxic.



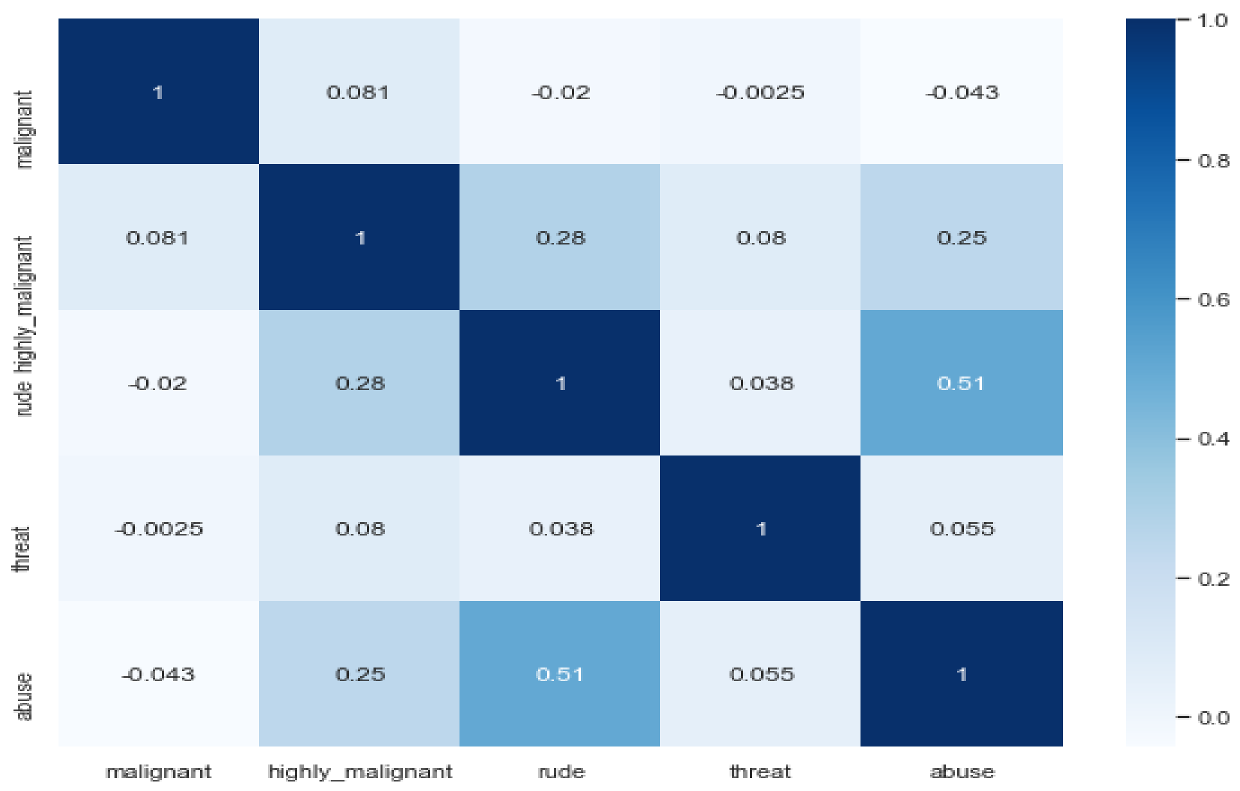
```
# example of clean comment
train.comment_text[0]
```

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

```
# example of toxic comment
train[train.malignant == 1].iloc[1,1]
```

```
'Hey... what is it..\n@ | talk .\nWhat is it... an exclusive group of some WP TALIBANS...who are good at destroying, self-appoi
nted purist who GANG UP any one who asks them questions abt their ANTI-SOCIAL and DESTRUCTIVE (non)-contribution at WP?\n\nAsk
Sityush to clean up his behavior than issue me nonsensical warnings...'
```

Correlation Matrix using Heatmap:

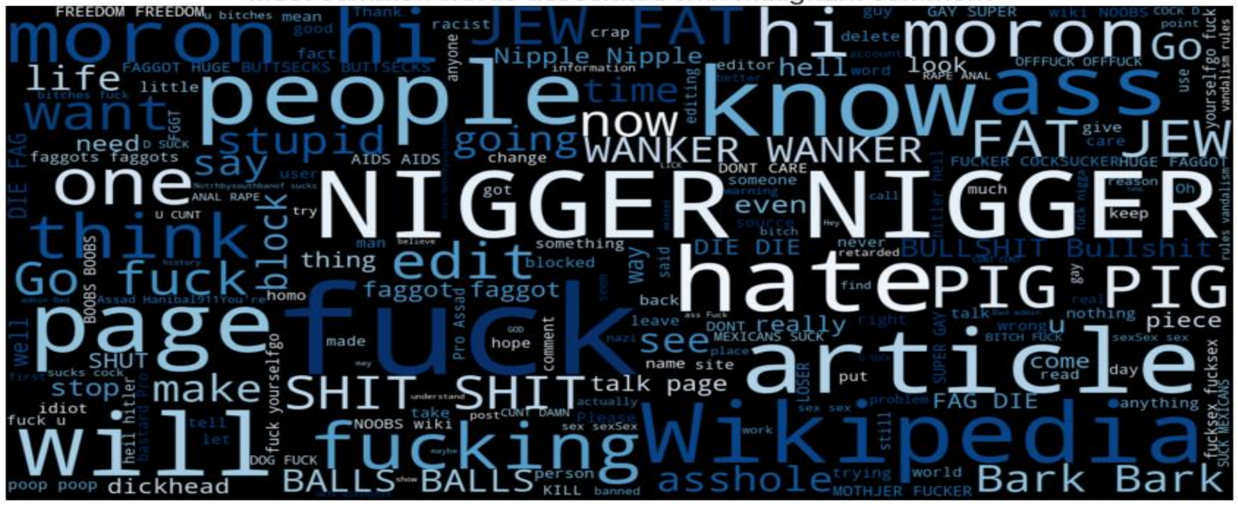


As seen in the cross-correlation matrix, there is a high chance of obscene comments to be insulting.

In order to get an idea of what are the words that contribute the most to different labels, we write a function to generate word clouds. The function takes in a parameter label (i.e., malignant, abuse, threat, etc.)

Using word cloud, importing all the words that are associated with malignant comments and there were quiet few words used for this classification.

Most common words assosiated with malignant comment

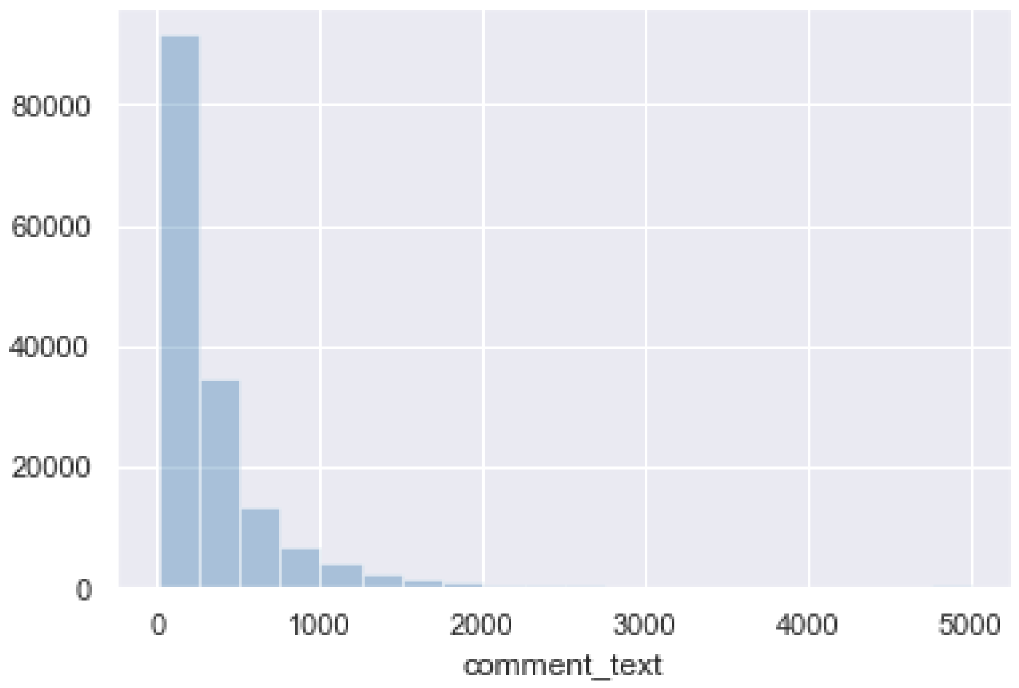


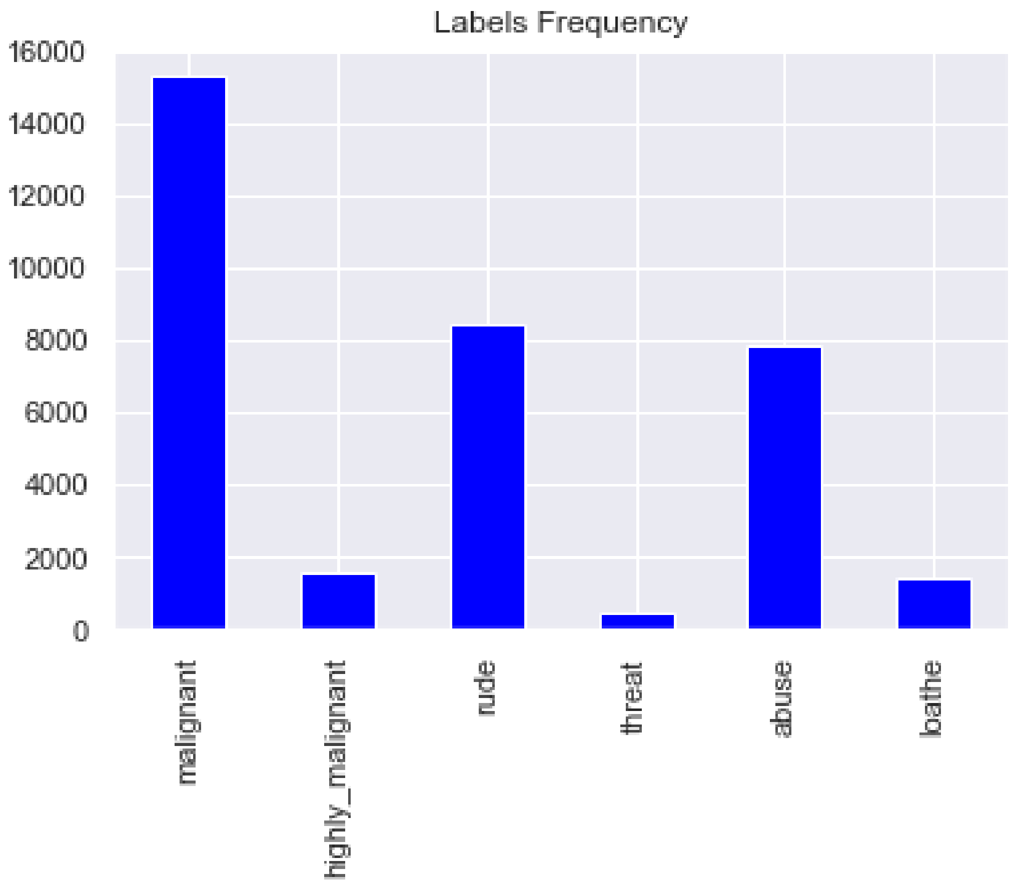
3. Methodology

3.1 Data Preprocessing

Data preprocessing refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models.

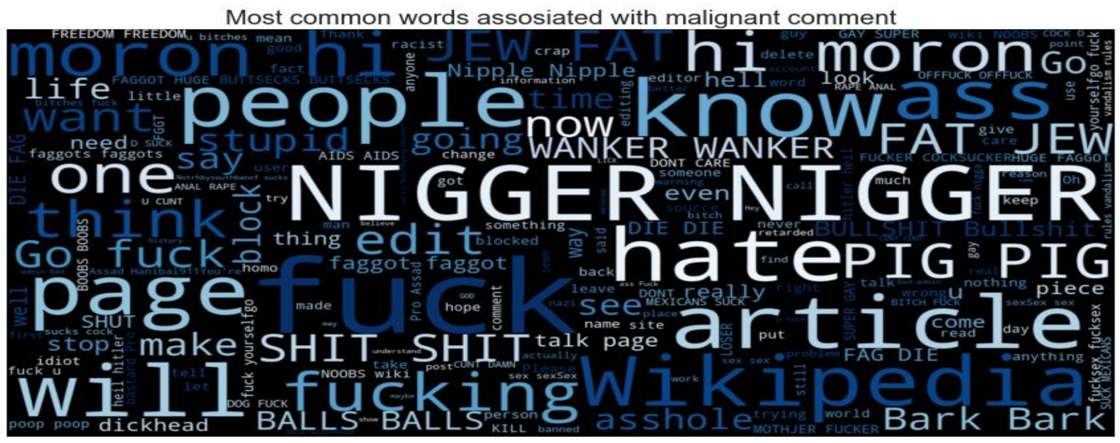
Since all of our data are text comments, we wrote our own tokenize () function, removing punctuations and special characters, stemming and/or lemmatizing the comments, and filtering out comments with length below 3. After benchmarking between different vectorizers (TFIDFVectorizer and CountVectorizer), we chose TFIDFVectorizer, which provides us with better performance.





The major concern of the data is that most of the comments are clean (i.e., non-malignant). There are only a few observations in the training data for Labels like threat. This indicates that we need to deal with imbalanced classes later on and indeed, we use different methods, such as resampling, choosing appropriate evaluation metrics, and choosing robust models to address this problem.

In order to get an idea of what are the words that contribute the most to different labels, we write a function to generate word clouds. The function takes in a parameter label (i.e., malignant, abuse, threat, etc.)



3.2 Stemming and Lemmatizing

1). 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".

2). **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.

```
test_labels = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
```

```
def tokenize(text):

    text = text.lower()
    regex = re.compile('[' + re.escape(string.punctuation) + '0-9\\r\\t\\n]')
    nopunct = regex.sub(" ", text)
    words = nopunct.split(' ')
    # remove any non ascii
    words = [word.encode('ascii', 'ignore').decode('ascii') for word in words]
    lmtzr = WordNetLemmatizer()
    words = [lmtzr.lemmatize(w) for w in words]
    words = [w for w in words if len(w) > 2]
    return words
```

3.3 Benchmarking Different Vectorizer

We determined to use TFIDF to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

Besides TFIDF, we also tried CountVectorizer. However, it is not performing as well as TFIDF. The TfidfVectorizer is actually CountVectorizer followed by TfidfTransformer. TfidfTransformer transforms a count matrix to a normalized Term-Frequency or TermFrequency-InverseDocumentFrequency representation. The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. That's why we can improve the accuracy here.

```
vector = TfidfVectorizer(ngram_range=(1, 1), analyzer='word',
                         tokenizer=tokenize, stop_words='english',
                         strip_accents='unicode', use_idf=1, min_df=10)
X_train = vector.fit_transform(train['comment_text'])
X_test = vector.transform(test['comment_text'])
```

After the transformation, we can take a look at some of the features below.

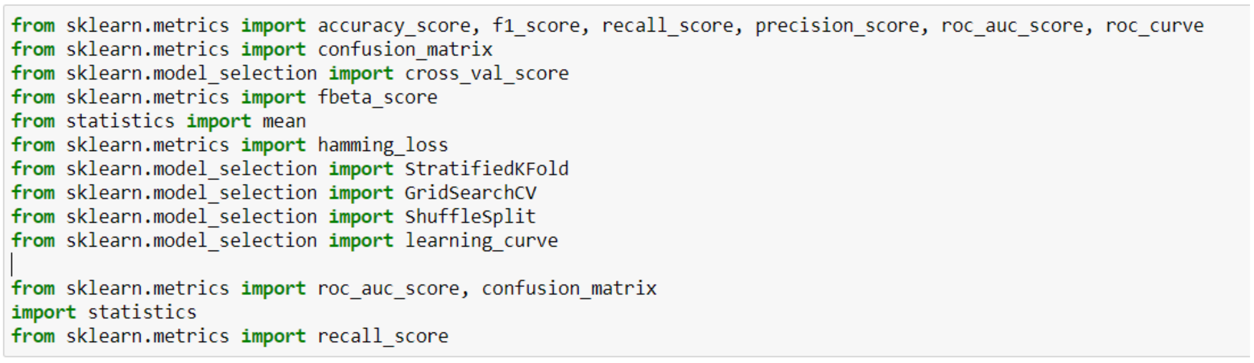
```
vector.get_feature_names()[0:20]
```

```
['aaa',
 'aap',
 'aardvark',
 'aaron',
 'aba',
 'abandon',
 'abandoned',
 'abandoning',
 'abandonment',
 'abbas',
 'abbey',
 'abbott',
 'abbreviated',
 'abbreviation',
 'abc',
 'abcnews',
 'abd',
 'abducted',
 'abduction',
 'abdul']
```

3.4 Implementation and Evaluation Metrics

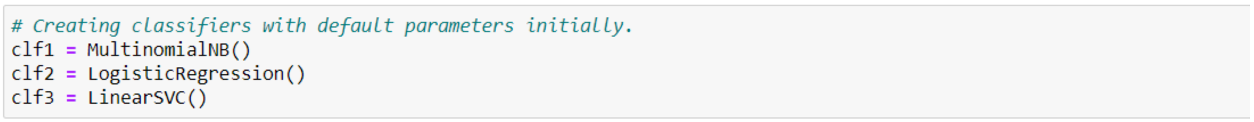
Our main metric for measuring model performance is F1-score, since we have 6 labels, the F1-score would be the average of 6 labels. We will also take other metrics into consideration while evaluating models, e.g., Hamming loss and recall.

We use cross Validation to compare between the baseline model and the other two models that we have chosen (LogisticRegression and LinearSVC).

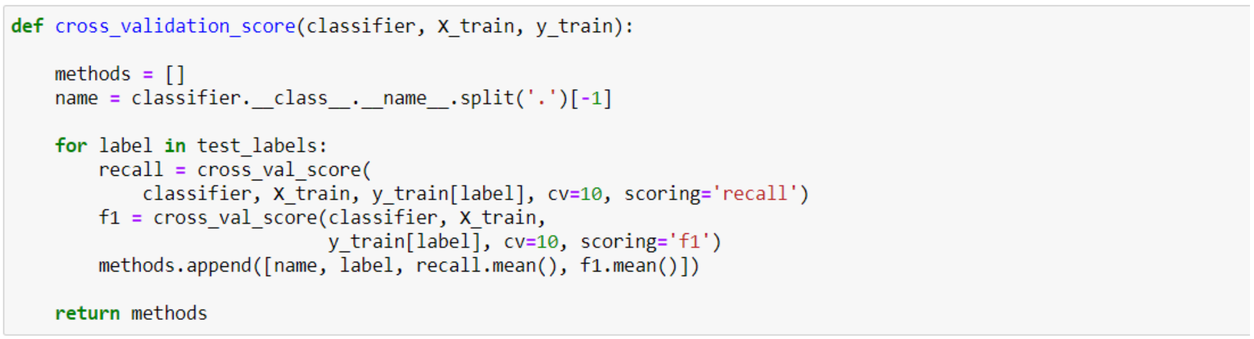


```python
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import fbeta_score
from statistics import mean
from sklearn.metrics 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
```



```python
# Creating classifiers with default parameters initially.
clf1 = MultinomialNB()
clf2 = LogisticRegression()
clf3 = LinearSVC()
```

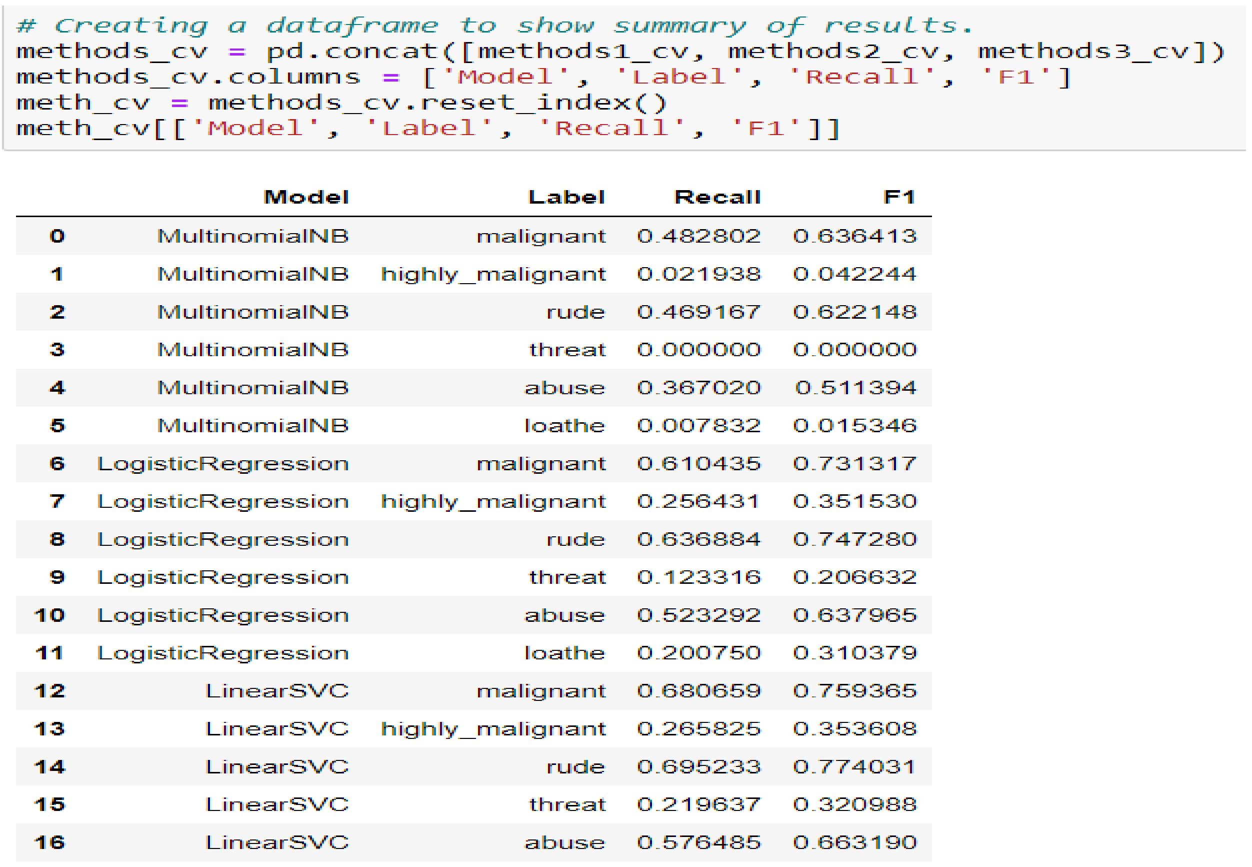


```python
def cross_validation_score(classifier, X_train, y_train):

    methods = []
    name = classifier.__class__.__name__.split('.')[-1]

    for label in test_labels:
        recall = cross_val_score(
            classifier, X_train, y_train[label], cv=10, scoring='recall')
        f1 = cross_val_score(classifier, X_train,
                             y_train[label], cv=10, scoring='f1')
        methods.append([name, label, recall.mean(), f1.mean()])

    return methods
```



```python
# Creating a dataframe to show summary of results.
methods_cv = pd.concat([methods1_cv, methods2_cv, methods3_cv])
methods_cv.columns = ['Model', 'Label', 'Recall', 'F1']
meth_cv = methods_cv.reset_index()
meth_cv[['Model', 'Label', 'Recall', 'F1']]
```

	Model	Label	Recall	F1
0	MultinomialNB	malignant	0.482802	0.636413
1	MultinomialNB	highly_malignant	0.021938	0.042244
2	MultinomialNB	rude	0.469167	0.622148
3	MultinomialNB	threat	0.000000	0.000000
4	MultinomialNB	abuse	0.367020	0.511394
5	MultinomialNB	loathe	0.007832	0.015346
6	LogisticRegression	malignant	0.610435	0.731317
7	LogisticRegression	highly_malignant	0.256431	0.351530
8	LogisticRegression	rude	0.636884	0.747280
9	LogisticRegression	threat	0.123316	0.206632
10	LogisticRegression	abuse	0.523292	0.637965
11	LogisticRegression	loathe	0.200750	0.310379
12	LinearSVC	malignant	0.680659	0.759365
13	LinearSVC	highly_malignant	0.265825	0.353608
14	LinearSVC	rude	0.695233	0.774031
15	LinearSVC	threat	0.219637	0.320988
16	LinearSVC	abuse	0.576485	0.663190

Based on the cross validation above, we noticed that overall, the linear SVC model and Logistic Regression model perform better. As a baseline model, Multinomial Naive Bayes does not perform well, especially for the threat label and loathe label because these two labels have the least number of observations.

3.5 Model Fitting

Evaluation Metrics Selection

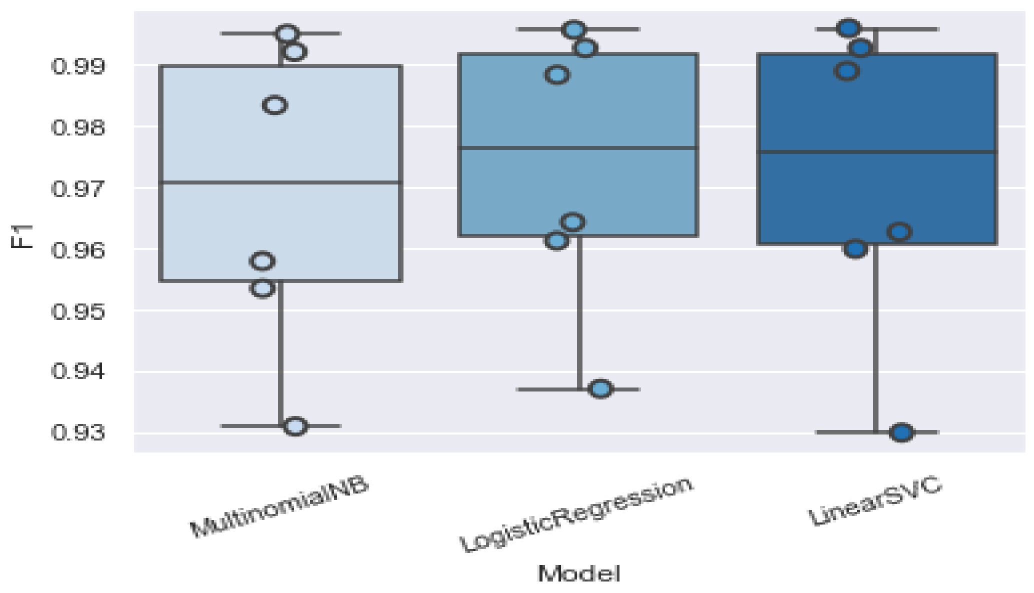
During the modeling process, we choose multiple different evaluation metrics to evaluate the performance of models based on the nature of our data:

- Recall
- F Score
- Hamming Loss

Basic Model Comparison

Using Multinomial Naive Bayes as our baseline model, we first used k-fold cross validation and compared the performance of the following three models without any hyperparameter tuning: Multinomial Naive Bayes, Logistic Regression, and Linear SVC. Logistic Regression and Linear SVC perform better than Multinomial Naive Bayes.

After checking how these models perform on the test data, we notice that Multinomial Naive Bayes does not perform as well as the other two models while Linear SVC in general out performs the others based on F1 score.



Overall, without any hyperparameter tuning, **LinearSVC** performs the best initially.

4. PipeLine

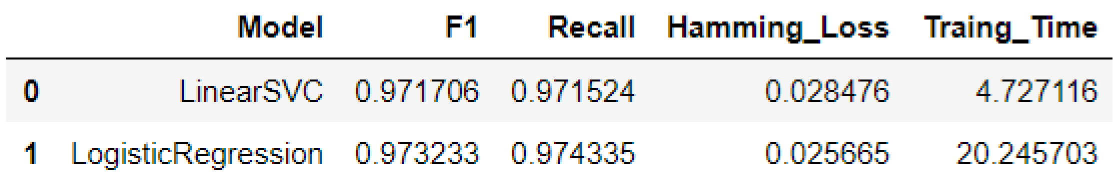
We have compared models without any hyperparameter tuning until now. Let's clean the code with pipeline and use some manually chosen hyperparameters to check how each model behaves. Since the greatest concern now is the imbalanced data, we decide to manually adjust class_weight for the models to see if we can achieve better results.

Since Logistic Regression and Linear SVM are performing better, we will focus on these two models. For display purpose, we will only include average F1 score, Recall, and Hamming Loss for comparison.

```
scores = pd.DataFrame(score_df,)
scores.columns = ['Model', 'F1', 'Recall', 'Hamming_Loss', 'Training_Time']
scores
```

4.1 Pipeline with Manual Hyperparameter Tuning

After accounting for the imbalanced data, the F1 score of Logistic Regression model has jumped to 0.9732 while Linear SVC has reached an average of 0.9717.

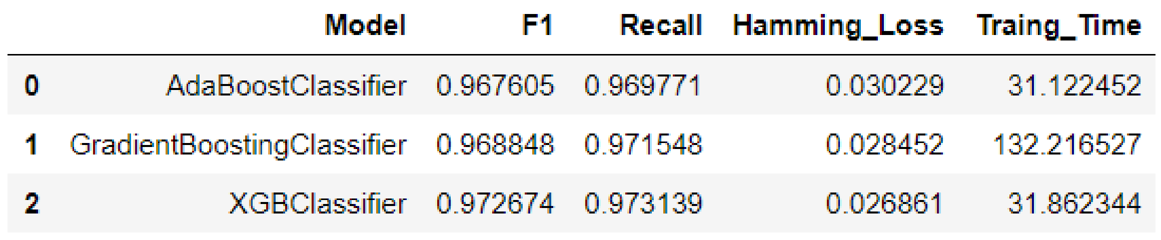


	Model	F1	Recall	Hamming_Loss	Traing_Time
0	LinearSVC	0.971706	0.971524	0.028476	4.727116
1	LogisticRegression	0.973233	0.974335	0.025665	20.245703

Notice that after adjusting class_weight, we are getting way better results than the basic models. LogisticRegression outperforms LinearSVC by approximately 1%.

4.2 Grid Search

With the help of grid search, we were able to find the "optimal" hyperparameters for the models and have reached an average of the best score of 0.9732 for Logistic Regression and 0.9717 for Linear SVC.

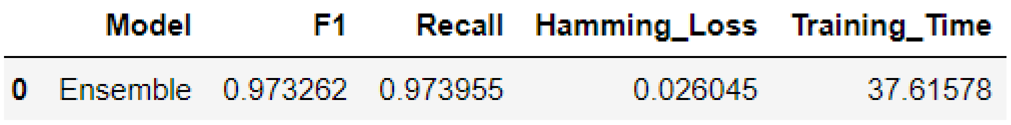


	Model	F1	Recall	Hamming_Loss	Traing_Time
0	AdaBoostClassifier	0.967605	0.969771	0.030229	31.122452
1	GradientBoostingClassifier	0.968848	0.971548	0.028452	132.216527
2	XGBClassifier	0.972674	0.973139	0.026861	31.862344

Since XGB outperforms other two boosting models, we decide to go ahead with XGB.

4.3 Ensembling

To ensemble different models, we firstly tried a few models based on tree boosting, then used a voting classier to ensemble one of the boosting model with the basic models in previous parts. We get a F1 score of 0.973262 and Hamming Loss of 0.026045 using Ensembling.

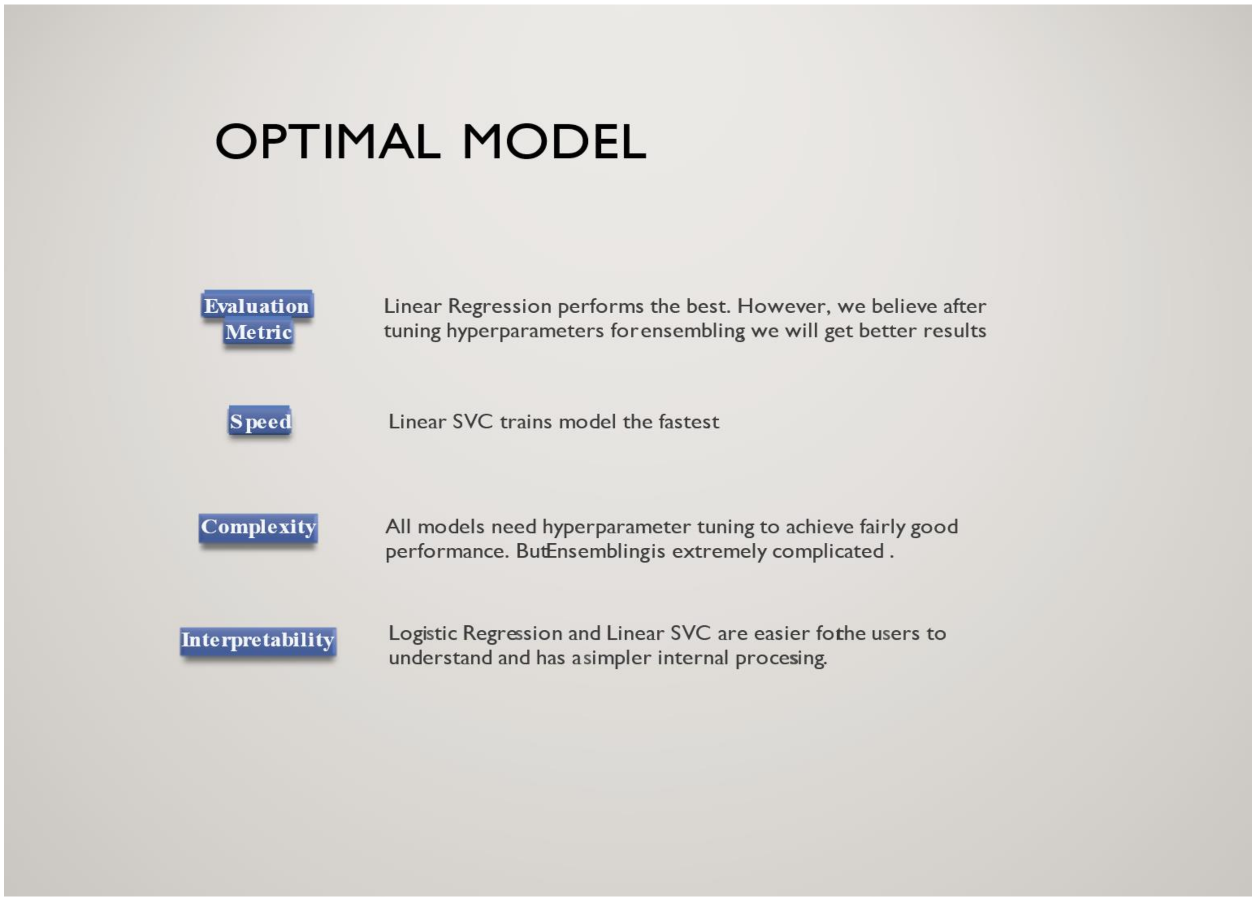


	Model	F1	Recall	Hamming_Loss	Training_Time
0	Ensemble	0.973262	0.973955	0.026045	37.61578

Ensembled model worked very well but still could not outperform LinearSVC since we did not tune the hyperparameters for the ensembled model.

Results

In terms of evaluation metric, **Logistic Regression** performs the best. But we believe after tuning hyperparameters for ensembling, we will get better results. Besides, Linear SVC trains model the fastest. Referring to interpretability, Logistic Regression is also easier for the users to understand and has a simpler internal processing. Therefore, we choose Logistic Regression as our optimal model.



OPTIMAL MODEL

Evaluation Metric	Linear Regression performs the best. However, we believe after tuning hyperparameters forensembling we will get better results
Speed	Linear SVC trains model the fastest
Complexity	All models need hyperparameter tuning to achieve fairly good performance. ButEnsemblingis extremely complicated .
Interpretability	Logistic Regression and Linear SVC are easier fothe users to understand and has asimpler internal procesing.

Future Scope

The current project predicts the type or toxicity in the comment. Here are the observations made throughout the project and features to be included in future:

- Analyze which age group is being malignant towards a particular group or brand.
- Add feature to automatically sensitize words which are classified as malignant.
- Automatically send alerts to the concerned authority if threats are classified as severe.
- Handle mistakes and short forms of words to get better accuracy of the result.