A project Report

On

Toxic comment classification

By:

Vivek kumar

21-05-2018

Contents

Introduction:

1.1 Problem Statement

The threat of abuse and harassment can be seen more often on online or social media platform that have numerous effect ranging from psychological to sociological. Many users restraining themselves to give any feedback on online platform and that can have big impact for businesses or any other sector that relies upon the users feedback for their services.

Basically comments are divided into three categories that is Positive, negative and neutral and negative categories are divided into different categories depending on their negativity scores like threat, obscene, insult, toxic ,identity hate, etc.

In this project we have given the task to prepare a multi label classification model that could be able to analyze and classify the Wikipedia's talk page into its respective categories

1.2 Data

Mentioned below are the sample of the Wikipedia's talk page data on which model has to be build. It contains the comment ID, comments along with its respective label like toxic, severe, etc.

Table 1.1 Document sample

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0000997932d777bf	Explanation Why 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	0	0	0	0	0	0
000103f0d9cfb60f	D'aww! He matches this background colour I'm seemingly stuck with. Thanks. (talk) 21:51, January 11, 2016 (UTC)	0	0	0	0	0	0

Table 1.2 Predictor variables.

Toxic	severe_toxic	obscene	threat	insult	identity_hate

Methodology

2.1 Basic Feature Extraction:

We can use data to extract number of features from it like the number of words, number of character, Average word length, Number of stopwords, Number of special character, Number of numeric, Number of upper case words. Using these analysis we get the information about the content of the document which could be helpful for the next steps that is data preprocessing.
Let's read the training dataset in order to perform different task on it.

2.1.1 Number of words:

The basic idea behind this is that negative comments contains lesser amount of words than a positive comments.

Table 2.1.1-Number of words count

	comment_text	word_count
0	Explanation\nWhy the edits made under my usern...	42
1	D'aww! He matches this background colour I'm s...	18
2	Hey man, I'm really not trying to edit war. It...	42
3	"\nMore\nI can't make any real suggestions on ...	112
4	You, sir, are my hero. Any chance you remember...	13

2.1.2 Number of characters:

The purpose is same as that of number of words.

Table 2.1.2 –Number of character counts

	comment_text	char_count
0	Explanation\nWhy the edits made under my usern...	264
1	D'aww! He matches this background colour I'm s...	112
2	Hey man, I'm really not trying to edit war. It...	233
3	"\nMore\nI can't make any real suggestions on ...	622
4	You, sir, are my hero. Any chance you remember...	67

2.1.3 Average word length-Average word length also helps in improving the models. It is obtained by taking the sum of all the words and divide it by the total length of the comments

	comment_text	avg_word
0	Explanation\nWhy the edits made under my usern...	5
1	D'aww! He matches this background colour I'm s...	5
2	Hey man, I'm really not trying to edit war. It...	4
3	"\nMore\nI can't make any real suggestions on ...	4
4	You, sir, are my hero. Any chance you remember...	4

2.1.4 Number of stopwords:

Stopwords don't gives extra information about the data so we remove those words from data and it also helps to reduce the size of the data.

Table 2.1.4 Count of stopwords

	comment_text	stopwords
0	Explanation\nWhy the edits made under my usern...	14
1	D'aww! He matches this background colour I'm s...	1
2	Hey man, I'm really not trying to edit war. It...	18
3	"\nMore\nI can't make any real suggestions on ...	49
4	You, sir, are my hero. Any chance you remember...	4

2.1.5 Number of numeric:

Numerical data does not have any usage in our model so it's better to remove these types of data if it is present in the data.

Table 2.1.5 Number of numerics

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

2.1.6 Number of uppercase words:

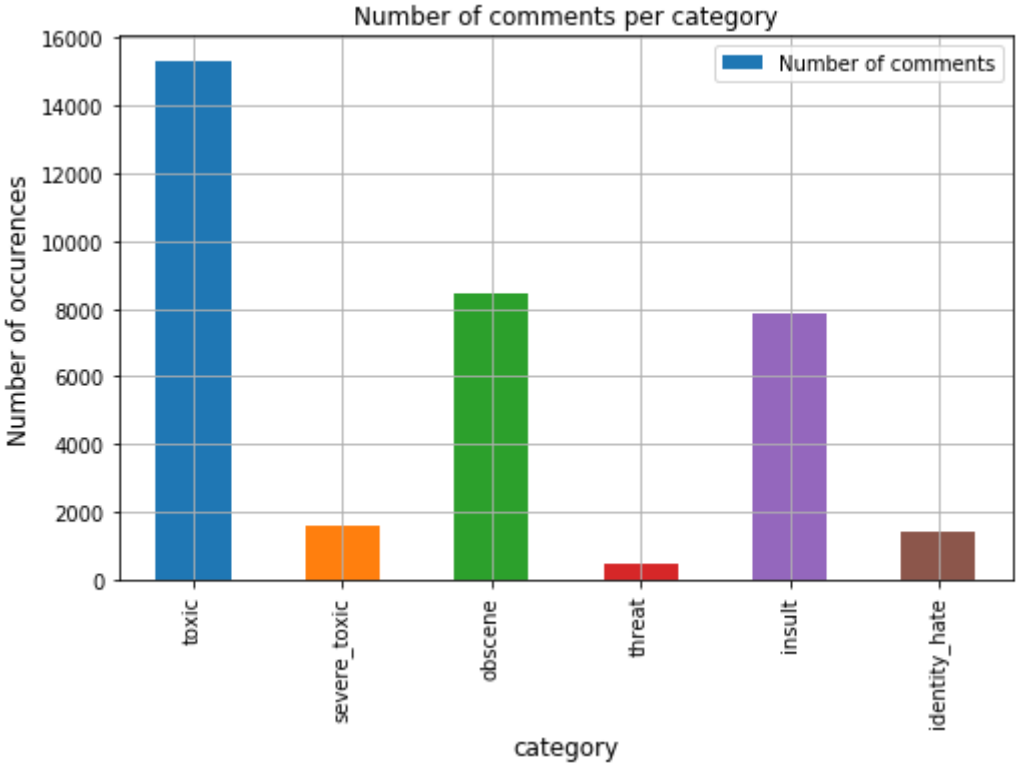
Upper case words used to express the expression like anger or rage. It also create additional features of the same words in term document matrix. So it become necessary to identify these types of words in the document.

	comment_text	upper
0	Explanation\nWhy the edits made under my usern...	2
1	D'aww! He matches this background colour I'm s...	1
2	Hey man, I'm really not trying to edit war. It...	0
3	"\nMore\nI can't make any real suggestions on ...	5
4	You, sir, are my hero. Any chance you remember...	0

2.1.7 Number of comment in each category:

Since count of negative comment if less than the count of positive comment , so we look into the data to check whether it hold true or not.

Fig 2.1.7 Number of comment in each category



2.2 Pre Processing

This process involves the preprocessing of raw data to prepare it for further analyses, basically in this process we transform the data to a required format so that the machine learning model could be built upon it.
So for better result it's better to convert the raw data into clean data because Machine learning model needs data in a specified format. Another purpose of cleaning data is that more than one algorithm could built upon it. It is used for extracting interesting and non-trivial knowledge from unstructured data .The objective of this study is to analyze the issue of preprocessing methods such as Tokenization, stopwords removal and stemming for the text document keywords.

Text documents are preprocessed because the text data contains number, dates, and the common words such as prepositions, articles etc. that are unlikely to give much information. It reduces data file size of the text documents and improve the efficiency and effectiveness of the model. One by one we will preprocess the data according to the model requirement.

2.2.1 Escaping HTML characters:

Web data often contain html content like < > &amp which creates noise in the data, so it is necessary to remove these entities from the data. Python htmlparser package are very helpful in accomplishing this task.

2.2.3 HTML decoding:

Though it carries some information but it can be ignored in our model.

Fig 2.2.3(a) before processing

```
df.comment_text[1504]
```

```
'Non-involved people most definitely cannot understand Yeshivish. A nice example of Yeshivish is the English Yated Neeman, whic
h can be found online at http://chareidi.shemayisrael.com . Many of their articles simply cannot be understood by a non-Jew or
 a non-Orthodox Jew. That is a simple fact. It is 99% English, but those few Hebrew (and Aramaic and Yiddish) words mixed in ma
ke many articles totally unreadable for the average person. I removed the disputed tag. It has been there for months and barely
 anything was done about the article since, so I presumed it was safe to remove it.  | (talk)'
```

Fig 2.2.3(b) after processing

```
df.comment_text[1504]
```

```
'Non-involved people most definitely cannot understand Yeshivish. A nice example of Yeshivish is the English Yated Neeman, whic
h can be found online at  . Many of their articles simply cannot be understood by a non-Jew or a non-Orthodox Jew. That is a si
mple fact. It is 99% English, but those few Hebrew (and Aramaic and Yiddish) words mixed in make many articles totally unreadab
le for the average person. I removed the disputed tag. It has been there for months and barely anything was done about the arti
cle since, so I presumed it was safe to remove it.  | (talk)'
```

2.2.4 Lower case:

This is the first step of our preprocessing task to convert all the words into lowercase words to avoid multiple copy of same words. As we can see from the Fig 2.2.1 that all the data gets converted into lower case.

Fig 2.2.4 Lower case data

```
df['comment_text'] = df['comment_text'].apply(lambda x: " ".join(x.lower() for x in x.split()))
df['comment_text'].head()
```

```
0    explanation why the edits made under my userna...
1    d'aww! he matches this background colour i'm s...
2    hey man, i'm really not trying to edit war. it...
3    " more i can't make any real suggestions on im...
4    you, sir, are my hero. any chance you remember...
```

2.2.5 Short cut words and removing extra space:

Short cut words creates noise in the data, so we will convert those data into their original form, we can clean those data using predefined function. From Fig 2.2.2 we can see that I'm gets converted to I am

```
def clean_text(text):
#     text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "can not ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text

df['comment_text'] = df['comment_text'].map(lambda com : clean_text(com))
df['comment_text'].head(5)
```

```
0    explanation why the edits made under my userna...
1    d aww he matches this background colour i am s...
2    hey man i am really not trying to edit war it ...
3    more i can not make any real suggestions on im...
4    you sir are my hero any chance you remember wh...
```

2.2.2 Removing Punctuation, stopwords & numeric data:

The next step is to remove punctuation, stopwords and numeric data from the train data as punctuation don't provide any useful information so it is better to remove punctuation as it will also helps to reduce the size of the training data.

Fig 2.2.6

```
stop=set(stopwords.words('english'))
exclude=set(string.punctuation)

# df_2_test['comment_text']=df_2_test['comment_text'].str.replace('[^\w\s]','')
df['comment_text']=df['comment_text'].str.replace('[^\w\s]','')

def clean(doc):
    stop_free=" ".join([i for i in doc.split() if i not in stop])
    punc_free=''.join(i for i in stop_free if i not in exclude )
    num_free=''.join(i for i in punc_free if not i.isdigit())
    return num_free
# df_2_test['comment_text']=[clean(df_2_test.iloc[i,1]) for i in range(0,df_2_test.shape[0])]
df['comment_text']=[clean(df.iloc[i,1]) for i in range(0,df.shape[0])]
```

```
df['comment_text'].head(10)
```

```
0    explanation edits made username hardcore metal...
1    aww matches background colour seemingly stuck ...
2    hey man really trying edit war guy constantly ...
3    make real suggestions improvement wondered sec...
4                      sir hero chance remember page
5            congratulations well use tools well talk
6                        cocksucker piss around work
7    vandalism matt shirvington article reverted pl...
8    sorry word nonsense offensive anyway intending...
9                 alignment subject contrary dulithgow
```

From **Fig 2.2.6** we can see that our data is almost cleaned, there is no numeric, stopwords and punctuation in the data

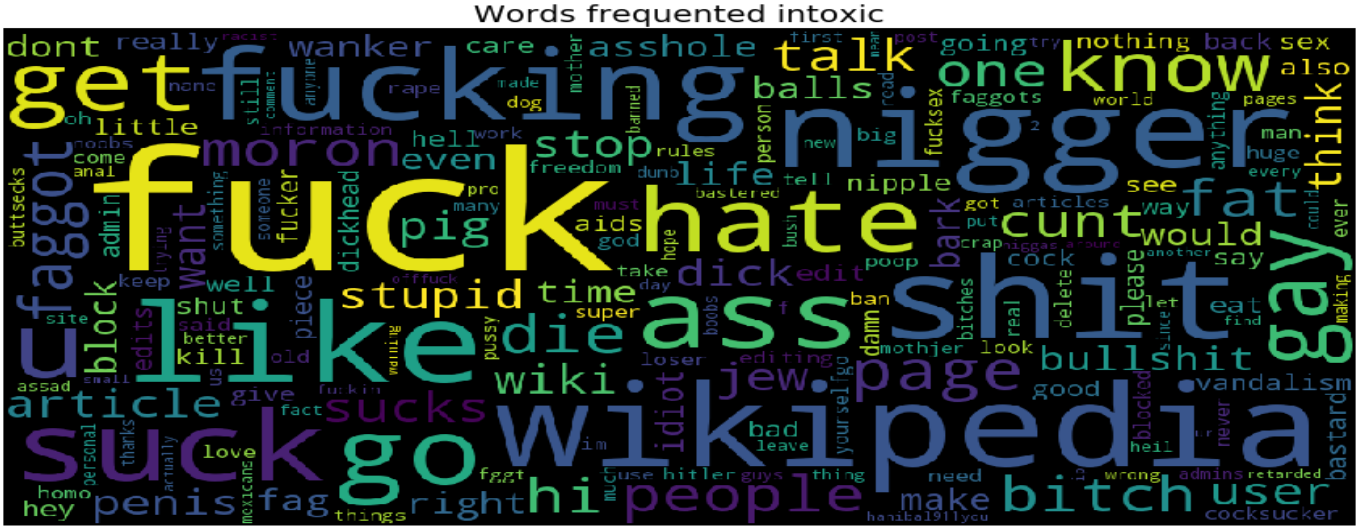
2.2.7 Lemmatization

Lemmatization is a more effective option than stemming because it converts the word into its root word, rather than just stripping the suffices. It makes use of the vocabulary and does a morphological analysis to obtain the root word. Therefore, we usually prefer using lemmatization over stemming.

3.Wordcloud:

Now our data is cleaned and ready for the advance processing. To get the understanding of the most frequent word in our data under different category we can take help of wordcloud package to create visual representation of the most frequent words.

Words frequented intoxic



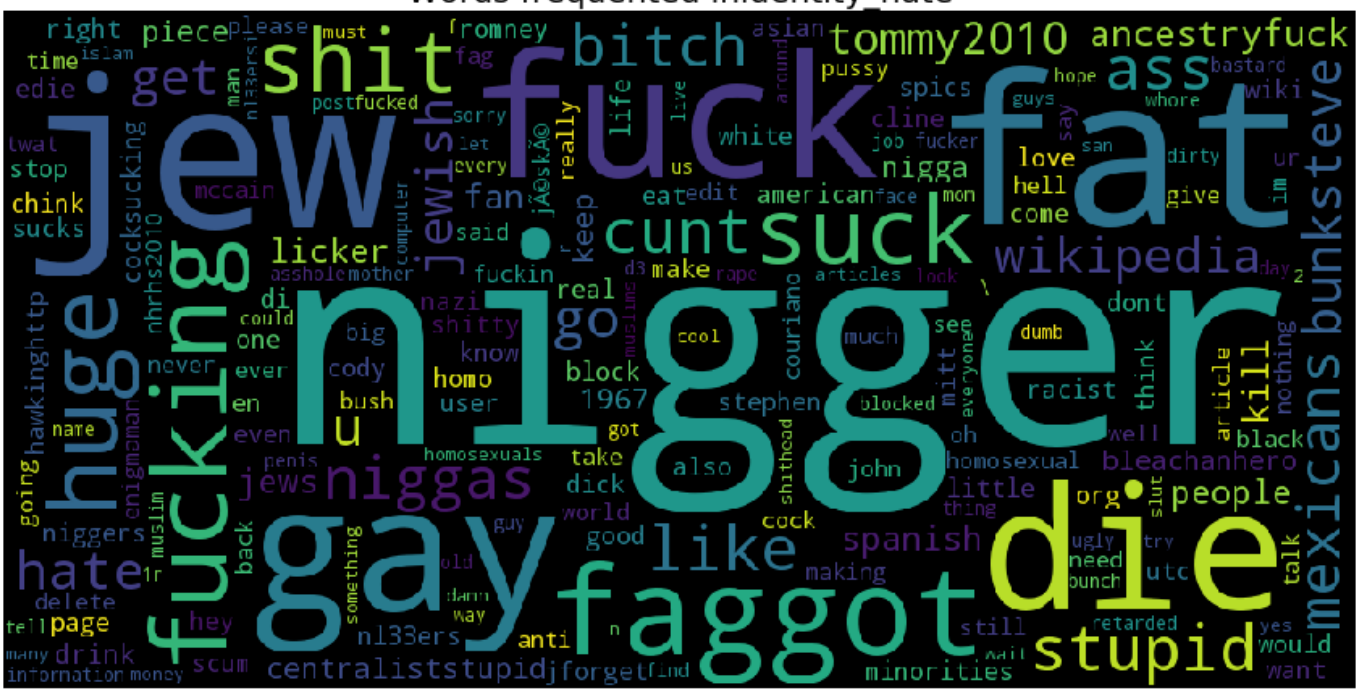
Words frequented insevere_toxic

Words frequented inobscene

Words frequented inthreat

Words frequented ininsult

Words frequented inidentity_hate



We have done Basic Preprocessing of data and now we will go for **advance processing** like tf-idf conversion of data and transforming into term document matrix so that it could be feed into machine learning algorithm.

4. Advance preprocessing.

4.1 Term frequency (Tf):

Term frequency is simply the ratio of the count of a word present in a sentence, to the length of the sentence.

Tf = (Number of times term T appears in the particular row) / (number of terms in that row).

```
# tf calculation
tf1 = (df_clean['comment_text'][0:5]).apply(lambda x: pd.value_counts(x.split(" "))).sum(axis = 0).reset_index()
tf1.columns = ['words','tf']
tf1.head(5)
```

	words	tf
0	accid	1.0
1	actual	1.0
2	appear	1.0
3	articl	1.0
4	background	1.0

4.2 Inverse document frequency (idf):

The idea behind this method is that some words occur more than one document frequently and that type words are not much important for our analysis therefore we use idf to penalize the occurrence of these words.
Idf= log (N/n),
where, N is the total number of rows and n is the number of rows in which the word was present.
The more the value of IDF, the more unique is the word.

Fig 4.2 idf-calculation

```
for i,word in enumerate(tf1['words']):
  tf1.loc[i, 'idf'] = np.log(df_clean.shape[0]/(len(df_clean[df_clean['comment_text'].str.contains(word)])))

tf1.head(5)
```

	words	tf	idf
0	match	1	5.225747
1	talk	1	1.609438
2	utc	1	3.091042
3	seem	1	2.833213
4	januari	1	5.075174

4.3 Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF is the multiplication of the TF and IDF which we calculated above**.**
tf-idf(d, t) = tf(t) * idf(d, t)
idf(d, t) = log [n / df(d, t)] + 1, The effect of adding "1" to the idf in the equation above is that terms with zero idf, i.e., terms that occur in all documents in a training set, will not be entirely ignored.

```
tf1['tfidf'] = tf1['tf'] * tf1['idf']
tf1
```

	words	tf	idf	tfidf
0	match	1	5.225747	5.225747
1	talk	1	1.609438	1.609438
2	utc	1	3.091042	3.091042
3	seem	1	2.833213	2.833213
4	januari	1	5.075174	5.075174

We can see that the TF-IDF has penalized words like ‘talk’ because they are commonly occurring words. However, it has given a high weight to “match” since that will be very useful in determining the category of the text.

4.4 N-grams:

N-gram is the combination of word together. The purpose behind this is to know the structure of the sentence like what word or letter are likely to follow other words or letter.
N=1, is called unigram, N=2 are bigram, N=3 are trigram and so on. The longer the N-gram, the more context it can capture but if N-gram is too long it will fail to capture the basic information of words.

```
# N-gram
TextBlob(df_clean['comment_text'][0]).ngrams(2)
```

```
[WordList([u'explanation', u'edits']),
 WordList([u'edits', u'made']),
 WordList([u'made', u'username']),
 WordList([u'username', u'hardcore']),
 WordList([u'hardcore', u'metallica']),
 WordList([u'metallica', u'fan']),
```

We don't have to calculate TF- IDF and N-gram every time beforehand and then multiply it to obtain TF-IDF. Instead, sklearn has a separate function called as tf-idf vectorizer which used to directly obtain N-gram and tf-idf:

4.5.Word to vector :

Sklearn Tf-idf vectorizer: In this method we will convert the word in a document in a vector form.

```
# tf-idf vectorizer
vectorizer=TfidfVectorizer(ngram_range=(1,3),
                           min_df=3,max_df=0.9,
                           strip_accents='unicode',
                            stop_words = 'english',
                            analyzer = 'word',
                           token_pattern=r'\w{1,}',
                            use_idf=1,
                           smooth_idf=1,
                            sublinear_tf=1,
                            max_features=50000)
```

In our model I have used mentioned above parameters in tf-idf function, where

ngram_range : tuple (min_n, max_n)=The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.

Min_df= ignores terms that have a document frequency (presence in % of documents) strictly lower than the given threshold.

Max_df=When building the vocabulary, it ignores terms that have a document frequency strictly higher than the given threshold. This could be used to exclude terms that are too frequent and are unlikely to help predict the label.

Max_features=Limit the amount of features (vocabulary) that the vectorizer will learn.

strip_accents : {'ascii', 'unicode', None}= Remove accents during the pre-processing step.'ascii' is a fast method that only works on characters that have an direct ASCII mapping. 'unicode' is a slightly slower method that works on any characters. None (default) does nothing.

analyzer : string, {'word', 'char', 'char_wb'} or callable= Whether the feature should be made of word or character n-grams.

use_idf : boolean, default=True ,Enable inverse-document-frequency reweighting.

smooth_idf : boolean, default=True Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.

sublinear_tf : boolean, default=False Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

Now we will apply tf-idf vectorizer to learn vocabulary and to calculate tf-idf of each feature present in a document and after that we will transform into a sparse matrix which consist of words as a feature in a sparse matrix form.

6. Modelling

6.1 Modelling and evaluation:

Now we have the data in proper format which we can use to feed it into machine learning algorithm. Our data is in form of matrix where all the words are a feature and the values are tf-idf that we have calculated earlier. Now we will build multiple machine learning model on top of this data and we will compare the accuracy score of each algorithm t determine which model we can use for multi-label classification. Since it is multi –label classification problem therefore we have to give input of the target variable in a sequence so that it could predict the target variable one by one.

First we will build a model using Naïve-Bayes algorithm because Naive Bayes classifiers, a family of classifiers that are based on the popular Bayes' probability theorem, are known for creating simple yet well performing models, especially in the fields of document classification and disease prediction.

Then we will build a model using logistic-regression algorithm since it performs very well when we have large amount of text data.

Then our last model will be a hybrid model using Naïve-Bayes log-count ratio as an additional feature as an input to logistic regression, this is known as NB-LR baseline and performs equally well like any other state of the art model.

6.1.1.Naive-Bayes. It performs very well short length of text data. We will apply this algorithm and check how it is performing on our data using different types of performance matrix.

Fig 6.1.1 sows the how to implement Naïve-Bayes Algorithm

```
NB_score=[]
NBauc_score=[]
for i,j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    modelNB=MultinomialNB()
    cv_score=np.mean(cross_val_score(modelNB,X_trainsplit_features,train_target,cv=5,scoring='accuracy'))
    NB_score.append(cv_score)
    modelNB.fit(X_trainsplit_features,train_target)
    y_pred_prob=modelNB.predict_proba(X_testsplit_features)[:,1]
    y_pred = modelNB.predict(X_testsplit_features)
    auc_score=metrics.roc_auc_score(y_test[j],y_pred_prob)
    NBauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(NBauc_score))
    print(classification_report(test_target, y_pred))
```

Accuracy of the model: **Fig 6.1.2**

	NB
toxic	0.937238
severe_toxic	0.987438
obscene	0.961379
threat	0.994491
insult	0.958040
identity_hate	0.988308

Classification Reports: It displays precision, recall,f1-score and support for the model in order to support easier interpretation and problem detection.

Precision:
Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as as the ratio of true positives to the sum of true and false positives. Said another way, "for all instances classified positive, what percent was correct?"

Recall:
Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. Said another way, "for all instances that were actually positive, what percent was classified correctly?"

F1 score:
The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

Support:
The support is the number of samples of the true response that lie in that class.

Fig 6.1.3

Toxic-Classification Report				
	precision	recall	f1-score	support
	0.94	1	0.97	47576
	0.91	0.4	0.56	5083
Avg/total	0.94	0.94	0.93	52659

Fig 6.1.4

Severe Toxic-Classification Report				
	precision	recall	f1-score	support
	0.99	1	0.99	52133
	0.05	0.02	0.03	526
Avg/total	0.98	0.99	0.98	52659

Fig 6.1.5

Obscene-Classification Report				
	precision	recall	f1-score	support
	0.96	1	0.98	49828
	0.83	0.36	0.5	2831
Avg/total	0.96	0.96	0.95	52659

Fig 6.1.6

Insult-Classification Report				
	precision	recall	f1-score	support
	0.99	1	0.99	52188
	0.03	0.01	0.02	471
Avg/total	0.98	0.99	0.99	52659

Fig 6.1.7

Insult-Classification Report				
	precision	recall	f1-score	support
	0.96	0.99	0.98	50016
	0.71	0.26	0.38	2643
Avg/total	0.95	0.96	0.95	52659

Fig 6.1.8

Threat-Classification Report				
	precision	recall	f1-score	support
	1	1	1	52507
	0.01	0.01	0.01	152
Avg/total	0.99	0.99	0.99	52659

We can see from Accuracy report and classification report that Naïve-Bayes is performing very well on the data set. Model has classified all the six categories with an accuracy more than 90% which is acceptable.

Now further we will look upon how Logistic Regression based models are performing on these data set.

6.2 Logistic Regression:

When we have to deal with high dimension data logistic regression outperforms tree based classification and text mining is such that area where we have high dimensional data and it performs exceptionally well on these data. We will now implement logistic regression algorithm to check whether it can perform well than Naïve-Bayes or not.

Below are the python codes that shows how we have implemented logistic regression in our data.

Fig 6.2.1

```
LR_score=[]
LRauc_score=[]
preds = np.zeros((len(df_2_test), len(categories)))

for i,j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    modelLR=LogisticRegression(C=12)
    cv_score=np.mean(cross_val_score(modelLR,X_trainsplit_features,train_target,cv=5,scoring='accuracy'))
    LR_score.append(cv_score)
    modelLR.fit(X_trainsplit_features,train_target)
    preds[:,i] = modelLR.predict_proba(Test_vector_csr.multiply(r))[:,1]
    y_pred_prob=modelLR.predict_proba(X_testsplit_features)[:,1]
    y_pred = modelLR.predict(X_testsplit_features)
    auc_score=metrics.roc_auc_score(y_test[j],y_pred_prob)
    LRauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(auc_score))
    print(classification_report(test_target, y_pred))
```

Accuracy report:

	NB	LR
toxic	0.937238	0.958564
severe_toxic	0.987438	0.989777
obscene	0.961379	0.978814
threat	0.994491	0.997185
insult	0.958040	0.970518
identity_hate	0.988308	0.992124

Classification report:

Fig 6.2.2

Toxic-Classification Report				
	precision	recall	f1-score	support
	0.96	0.99	0.98	47576
	0.9	0.64	0.75	5083
Avg/total	0.96	0.96	0.96	52659

Fig 6.2.3

Severe Toxic-Classification Report				
	precision	recall	f1-score	support
	0.99	1	1	52133
	0.54	0.25	0.34	526
Avg/total	0.99	0.99	0.99	52659

Obscene-Classification Report				
	precision	recall	f1-score	support
	0.98	1	0.99	49828
	0.91	0.66	0.77	2831
Avg/total	0.98	0.98	0.98	52659

Threat-Classification Report				
	precision	recall	f1-score	support
	1	1	1	52507
	1	1	1	52659
Avg/total	0.98	0.98	0.98	52659

Fig 6.2.6

Insult-Classification Report				
	precision	recall	f1-score	support
	0.98	0.99	0.98	50016
	0.81	0.54	0.64	2643
Avg/total	0.97	0.97	0.97	52659

Fig 6.2.7

Identity Hate-Classification Report				
	precision	recall	f1-score	support
	0.99	1	1	52188
	0.69	0.22	0.34	471
Avg/total	0.99	0.99	0.99	52659

As we can see from accuracy and classification report of Logistic Regression model, it has outperformed Naïve-Bayes algorithm in every aspect. This algorithm not only maintained accuracy but also maintained high accuracy of other classification parameter as well.

6.3 NB-LR Baseline

Now proceeding to our next model that is NB-LR baseline that works Naïve-Bayes log-count ratio as an additional feature as an input to logistic regression model, this is known as NB-LR baseline and performs equally well like any other state of the art model.

Log count ratio (r) =Ratio of feature ‘f’ in document (with label=1)/Ratio of feature ‘f’ in document (with label=0).

Where,

Ratio of feature ‘f’ in document (with label=1) =Number of times a document (with label=1) contains a feature ‘f’ divided by the number of document (with label=1)
Ratio of feature ‘f’ in document (with label=0) =Number of times a document (with label=0) contains a feature ‘f’ divided by the number of document (with label=0)

Mentioned Below are the basic Naïve-Bayes feature extraction function.

```
def pr(y_i, y):
    p = x[y==y_i].sum(0)
    return (p+1) / ((y==y_i).sum()+1)
```

Where,
Y_i = Label
y = Multi label categories in the training data
x=Training data in matrix form.
Python codes to implement NB-LR baseline:

```
NBLR_score=[]
NBLRauc_score=[]
for i, j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    print('fit', j)
    y = y_train[j].values
    r = np.log(pr(1,y) / pr(0,y))
    m = LogisticRegression(C=3,dual=True)
    x_nb = x.multiply(r)
    model=m.fit(x_nb,y_train[j])
#     preds[:,i] = modelLR.predict_proba(Test_vector_csr.multiply(r))[:,1]
    y_pred_prob=m.predict_proba(X_testsplit_features)[:,1]
    y_pred = m.predict(X_testsplit_features)
    cv_score=np.mean(cross_val_score(m,x_nb,train_target,cv=5,scoring='accuracy'))
    NBLR_score.append(cv_score)
    auc_score=metrics.roc_auc_score(y_test[j], y_pred_prob)
    NBLRauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(auc_score))
    print(classification_report(test_target, y_pred))
```

Accuracy Report:

	NB	LR	NB-LR
toxic	0.937238	0.958564	0.961342
severe_toxic	0.987438	0.989777	0.989627
obscene	0.961379	0.978814	0.980096
threat	0.994491	0.997185	0.997138
insult	0.958040	0.970518	0.971182
identity_hate	0.988308	0.992124	0.991928

Classification Reports:

Fig 6.3.1

Toxic-Classification Report				
	precision	recall	f1-score	support
	0.93	0.9	0.92	47576
	0.29	0.37	0.32	5083
Avg/total	0.87	0.85	0.86	52659

Fig 6.3.2

Severe Toxic-Classification Report				
	precision	recall	f1-score	support
	0.99	1	0.99	52133
	0	0	0	526
Avg/total	0.98	0.99	0.99	52659

Fig 6.3.4

Obscene-Classification Report				
	precision	recall	f1-score	support
	0.95	1	0.97	49828
	0.96	0.03	0.07	2831
Avg/total	0.95	0.95	0.92	52659

Fig 6.3.5

Threat-Classification Report				
	precision	recall	f1-score	support
	1	1	1	52507
	0	0	0	152
Avg/total	0.99	1	1	52659

Fig 6.3.6

Insult-Classification Report				
	precision	recall	f1-score	support
	0.95	1	0.97	50016
	0.71	0.02	0.04	2643
Avg/total	0.94	0.95	0.93	52659

Fig 6.3.7

Identity Hate-Classification Report				
	precision	recall	f1-score	support
	0.99	1	1	52188
	0	0	0	471
Avg/total	0.98	0.99	0.99	52659

We can see from accuracy report that NB-LR is performing well in three of the categories and it performing almost equally on the other remaining two categories.

7. Conclusion:

Model selection

We have built three model and tested the accuracy of all the model now we will analyse which model we should select for the deployment.

Comparison on the basis of Accuracy:

Accuracy of all the models are above 90% hence all the models. Acceptable in deployment but LR and NB-LR has an edge Over NB.

Now we will analyse into the classification reports of all the models:

Comparing all the classification reports of all the models we can say that:
In class toxic LR is performing well than NB-LR and NB.
In class severe_toxic LR is performing better than all
In class obscene LR outperforms all other models
In class Threat NB-LR is doing well than other two models.
In class Insult LR is marginally good over other models
In class Identity hate LR is better than other two models

Now we have analysed accuracy and classification reports of all the models therefore we can select either LR or NB-LR because both the model are performing outstanding on the text dataset with accuracy of more than 95%. In our project I am selecting NB-LR model.

Appendix B - Python Code

Data visualisation before cleaning (From fig 2.1.1 to 2.2.6)

```
# Number of comments
df_toxic=df.drop(['id','comment_text'],axis=1)
counts=[]
categories=list(df_toxic.columns.values)
for i in categories:
   counts.append((i,df_toxic[i].sum()))

df_stats=pd.DataFrame(counts,columns=['category','Number of comments'])
df_stats

df_stats.plot(x='category',y='Number of comments',kind='bar',legend='False',grid=True,figsize=(8,5))
plt.title("Number of comments per category")
plt.ylabel('Number of occurences',fontsize=12)
plt.xlabel('category',fontsize=12)
```

```
# Number of words
df['word_count'] = df['comment_text'].apply(lambda x: len(str(x).split(" ")))
df[['comment_text','word_count']].head()
```

```
# Average word length:
def avg_word(sentence):
  words = sentence.split()
  return (sum(len(word) for word in words)/len(words))

df['avg_word'] = df['comment_text'].apply(lambda x: avg_word(x))
df[['comment_text','avg_word']].head()
```

```
# Number of stopwords
df['stopwords'] = df['comment_text'].apply(lambda x: len([x for x in x.split() if x in stop]))
df[['comment_text','stopwords']].head()
```

```
# Number of numeric
df['numerics'] = df['comment_text'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))
df[['comment_text','numerics']].head()
```

```
# number of uppercase words
df['upper'] = df['comment_text'].apply(lambda x: len([x for x in x.split() if x.isupper()]))
df[['comment_text','upper']].head()
```

Data Cleaning:

```
def clean_text(text):
    text = text.lower()
#     text = re.sub('https?://[A-za-z0-9./]+','',text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "can not ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text
```

```
df_2_test['comment_text'] = df_2_test['comment_text'].map(lambda com : clean_text(com))
df['comment_text'] = df['comment_text'].map(lambda com : clean_text(com))
df['comment_text'].head(5)
```

```
stop=set(stopwords.words('english'))
exclude=set(string.punctuation)

df_2_test['comment_text']=df_2_test['comment_text'].str.replace('[^\w\s]','')
df['comment_text']=df['comment_text'].str.replace('[^\w\s]','')

def clean(doc):
    stop_free=" ".join([i for i in doc.split() if i not in stop])
    punc_free=''.join(i for i in stop_free if i not in exclude )
    num_free=''.join(i for i in punc_free if not i.isdigit())
    return num_free
df_2_test['comment_text']=[clean(df_2_test.iloc[i,1]) for i in range(0,df_2_test.shape[0])]
df['comment_text']=[clean(df.iloc[i,1]) for i in range(0,df.shape[0])]
```

```
from nltk.stem import WordNetLemmatizer
wlem = WordNetLemmatizer()
```

```
df_clean['comment_text'] = df_clean['comment_text'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))
df_clean['comment_text'].head()
```

Wordcloud:

```
target = ['toxic','severe_toxic','obscene','threat','insult','identity_hate']
toxic=df[df.toxic==1]['comment_text'].values
severe_toxic=df[df.severe_toxic==1]['comment_text'].values
obscene=df[df.obscene==1]['comment_text'].values
threat=df[df.threat==1]['comment_text'].values
insult=df[df.insult==1]['comment_text'].values
identity_hate=df[df.identity_hate==1]['comment_text'].values

word_counter = {}
stop=set(stopwords.words('english'))
def clean_text(text):
    stop=set(stopwords.words('english'))
    text = re.sub('[{}]'.format(string.punctuation), ' ', text.lower())
    return ' '.join([word for word in text.split() if word not in (stop)])

for categ in target:
  d = Counter()
  df[df[categ] == 1]['comment_text'].apply(lambda t: d.update(clean_text(t).split()))
  word_counter[categ] = pd.DataFrame.from_dict(d, orient='index')\
                                        .rename(columns={0: 'count'})\
                                        .sort_values('count', ascending=False)
```

```
word_counter = {}

def clean_text(text):
    text = re.sub('[{}]'.format(string.punctuation), ' ', text.lower())
    return ' '.join([word for word in text.split() if word not in (stop)])

for categ in target:
  d = Counter()
  df[df[categ] == 1]['comment_text'].apply(lambda t: d.update(clean_text(t).split()))
  word_counter[categ] = pd.DataFrame.from_dict(d, orient='index')\
                                        .rename(columns={0: 'count'})\
                                        .sort_values('count', ascending=False)
for categ in target:
    print(categ)
    wc = word_counter[categ]
    wordcloud = WordCloud(width = 1000, height = 500,stopwords=STOPWORDS, background_color = 'black').generate_from_frequencies(
                        wc.to_dict()['count'])
    plt.figure(figsize = (15,8))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title("Words frequented in"+str(categ), fontsize=20)
    plt.show()
```

Advance pre-processing:

```
# Advance preprocessing
```

```
tf calculation
tf1 = (df_clean['comment_text'][0:1]).apply(lambda x: pd.value_counts(x.split(" "))).sum(axis = 0).reset_index()
tf1.columns = ['words','tf']
tf1.head(20)
```

```
for i,word in enumerate(tf1['words']):
  tf1.loc[i, 'idf'] = np.log(df_clean.shape[0]/(len(df_clean[df_clean['comment_text'].str.contains(word)])))

tf1.head(5)
```

```
tf1['tfidf'] = tf1['tf'] * tf1['idf']
tf1
```

```
# tf-idf vectorizer
word_vectorizer=TfidfVectorizer(ngram_range=(1,4),
                            min_df=3,max_df=0.9,
                            strip_accents='unicode',
                             stop_words = 'english',
                             analyzer = 'word',
                            token_pattern=r'\w{1,}',
                             use_idf=1,
                            smooth_idf=1,
                             sublinear_tf=1,
                             max_features=50000)
vectorizer = word_vectorizer
```

```
vectorizer.fit(df['comment_text'])
```

```
Train_vector_csr=vectorizer.transform(df.comment_text)
Test_vector_csr=vectorizer.transform(df_2_test.comment_text)
```

```
Train_all_text=Train_vector_csr
```

Splitting data as train and test and transforming into matrix:

```
x1=df['comment_text']
y1=df.iloc[:,2:8]
```

```
X_train, X_test,y_train,y_test= train_test_split(x1,y1,random_state=42, test_size=0.33)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
vectorizer.fit(X_train)
X_trainsplit_features=vectorizer.transform(X_train)
```

```
X_testsplit_features=vectorizer.transform(X_test)
```

```
categories = ['toxic','severe_toxic','obscene','threat','insult','identity_hate']
```

Naïve-Bayes Model:

```
# NB- model completed:
NB_score=[]
NBauc_score=[]
for i,j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    modelNB=MultinomialNB()
    cv_score=np.mean(cross_val_score(modelNB,X_trainsplit_features,train_target,cv=5,scoring='accuracy'))
    NB_score.append(cv_score)
    modelNB.fit(X_trainsplit_features,train_target)
    y_pred_prob=modelNB.predict_proba(X_testsplit_features)[:,1]
    y_pred = modelNB.predict(X_testsplit_features)
    auc_score=metrics.roc_auc_score(y_test[j],y_pred_prob)
    NBauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(NBauc_score))
    print(classification_report(test_target, y_pred))
```

```
Accuracy_score=pd.DataFrame(index=categories)
Accuracy_score['NB']=NB_score
Accuracy_score
```

Logistic-regression Model:

```
# LR model completed
LR_score=[]
LRauc_score=[]


for i,j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    modelLR=LogisticRegression(C=12)
    cv_score=np.mean(cross_val_score(modelLR,X_trainsplit_features,train_target,cv=5,scoring='accuracy'))
    LR_score.append(cv_score)
    modelLR.fit(X_trainsplit_features,train_target)
    y_pred_prob=modelLR.predict_proba(X_testsplit_features)[:,1]
    y_pred = modelLR.predict(X_testsplit_features)
    auc_score=metrics.roc_auc_score(y_test[j],y_pred_prob)
    LRauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(auc_score))
    print(classification_report(test_target, y_pred))
```

```
Accuracy_score['LR']=LR_score
Accuracy_score
```

NB-LR Baseline Model:

```
# NB-LR model completed:
x=X_trainsplit_features
def pr(y_i, y):
    p = x[y==y_i].sum(0)
    return (p+1) / ((y==y_i).sum()+1)

def get_mdl(y):
    y = y.values
    r = np.log(pr(1,y) / pr(0,y))
    m = LogisticRegression(C=3)
    x_nb = x.multiply(r)
    return m.fit(x_nb, y), r
```

```
# preds = np.zeros((len(df_2_test), len(categories)))
NBLR_score=[]
NBLRauc_score=[]
for i, j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    print('fit', j)
    y = y_train[j].values
    r = np.log(pr(1,y) / pr(0,y))
    m = LogisticRegression(C=3,dual=True)
    x_nb = x.multiply(r)
    model=m.fit(x_nb,y_train[j])
#     preds[:,i] = modelLR.predict_proba(Test_vector_csr.multiply(r))[:,1]
    y_pred_prob=m.predict_proba(X_testsplit_features)[:,1]
    y_pred = m.predict(X_testsplit_features)
    cv_score=np.mean(cross_val_score(m,x_nb,train_target,cv=5,scoring='accuracy'))
    NBLR_score.append(cv_score)
    auc_score=metrics.roc_auc_score(y_test[j], y_pred_prob)
    NBLRauc_score.append(auc_score)
    print("Accuracy score for class {} is {}".format(j,cv_score))
    print("CV ROC_AUC score {}\n".format(auc_score))
    print(classification_report(test_target, y_pred))
```

Model submission:

```
# Final submissiom
preds = np.zeros((len(df_2_test), len(categories)))
NBLR_score=[]
NBLRauc_score=[]
for i, j in enumerate(categories):
    train_target=y_train[j]
    test_target=y_test[j]
    print('fit', j)
    m,r = get_mdl(df[j])
    preds[:,i] = m.predict_proba(test_x.multiply(r))[:,1]
# #     y_pred_prob=m.predict_proba(X_testsplit_features)[:,1]
# #     y_pred = m.predict(X_testsplit_features)
# #     cv_score=np.mean(cross_val_score(m,X_trainsplit_features,train_target,cv=5,scoring='accuracy'))
#     NBLR_score.append(cv_score)
#     auc_score=metrics.roc_auc_score(y_test[j], y_pred_prob)
#     NBLRauc_score.append(auc_score)
#     print("Accuracy score for class {} is {}".format(j,cv_score))
#     print("CV ROC_AUC score {}\n".format(auc_score))
#     print(classification_report(test_target, y_pred))
#     And finally, create the submission file.


    submid = pd.DataFrame({'id': subm["id"]})
    submission = pd.concat([submid, pd.DataFrame(preds, columns = categories)], axis=1)
    submission.to_csv('submission.csv', index=False)
```

References

https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/

https://www.wikipedia.com.

https://www.kaggle.com/abhi111/naive-bayes-baseline-and-logistic-regression

Baselines and Bigrams: Simple, Good Sentiment and Topic Classification [Sida Wang and Christopher D. Manning].