Toxic Comments

Sai Venkatesh, George Abraham, Pratyush Nandi

IMT2017012, IMT2017019, IMT2017518

November 2019

1 Problem Statement

Given a comment we are required to classify it into different types of toxicity like

- toxic
- severe toxic
- threat,
- obscenity,
- insults,
- identity hate.

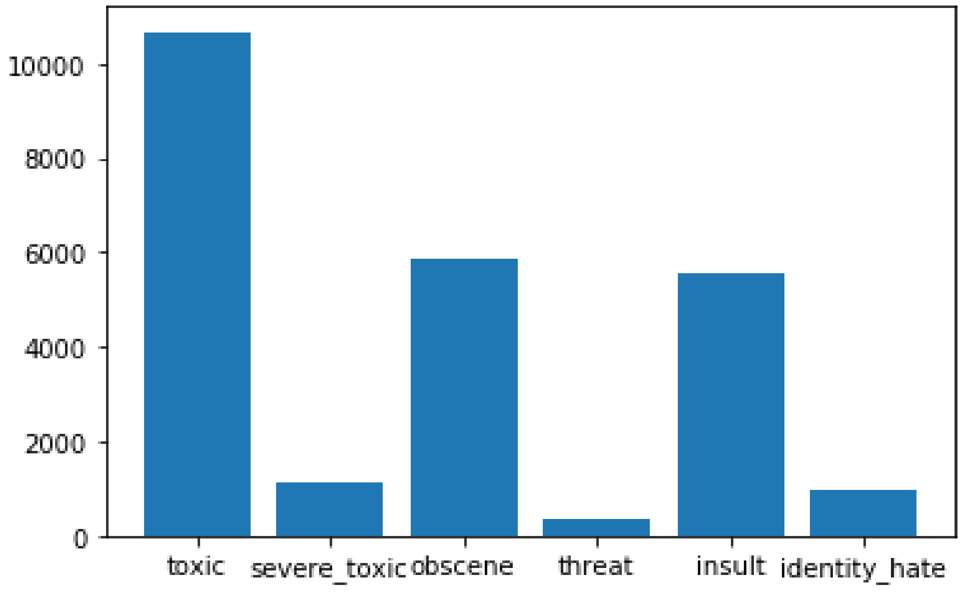
To achieve this we require a multi-label classifier where data can belong to multiple labels at the same time.

2 Format

The given Data set has 7 columns, in with one column consists comments which are taken from an online platform.

2.1 Characteristics of the labels

The no. of comments for each label type is shown in the graph below:



As we can see

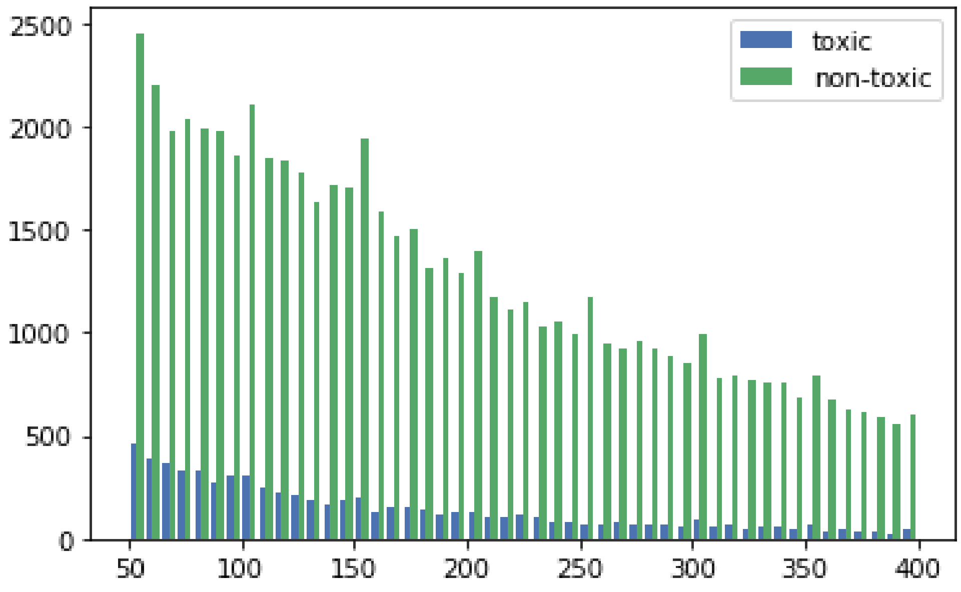
- The fraction of **toxic** comments is: **0.0955**
- The fraction of **severe toxic** labeled comments is: **0.0101**
- The fraction of **obscene** labeled comments is: **0.0526**
- The fraction of **threat** labeled comments is: **0.0031**
- The fraction of **insult** labeled comments is: **0.0495**
- The fraction of **identity hate** labeled comments is: **0.0088**

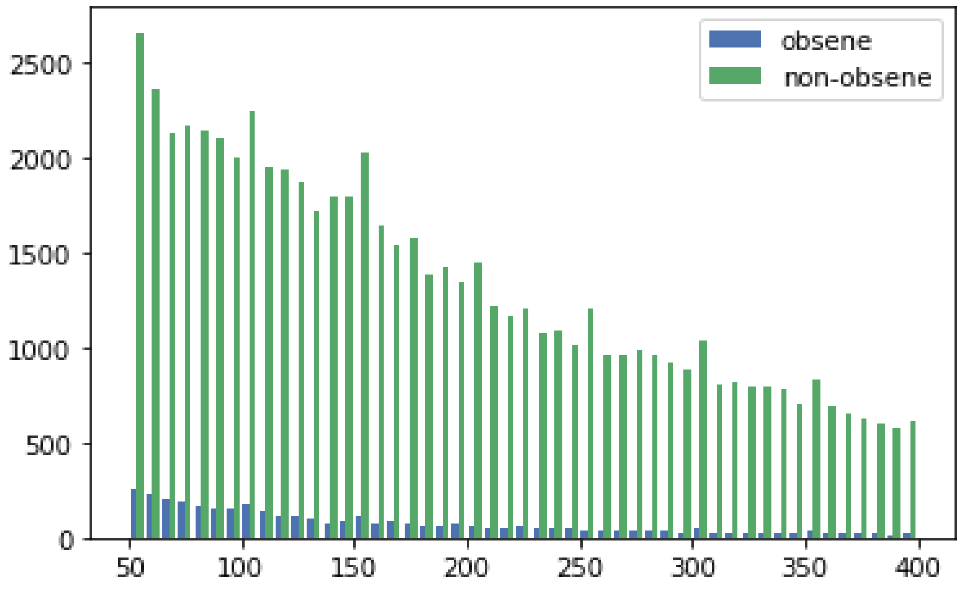
As we can see,the labels **threat** and **identity hate** are very rare and require a different approach than the other labels.

2.2 Exploratory data analysis

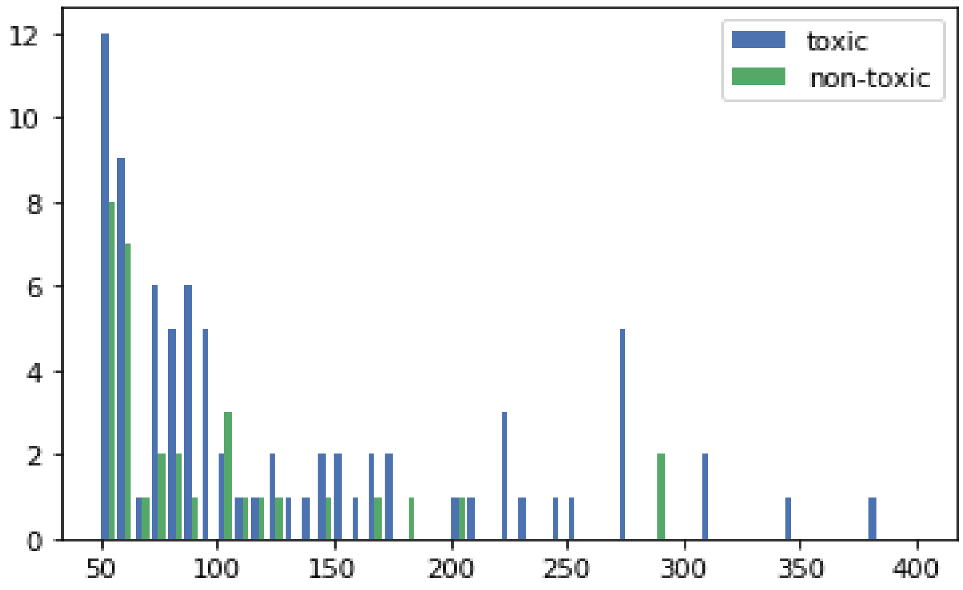
The given string (comment text) can have multiple features like :

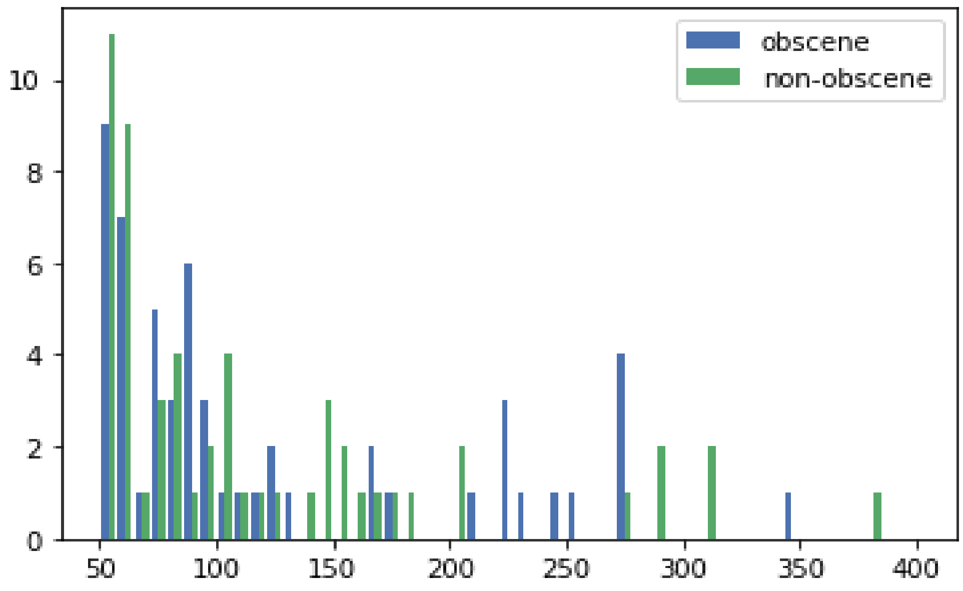
2.2.1 Length of the comment



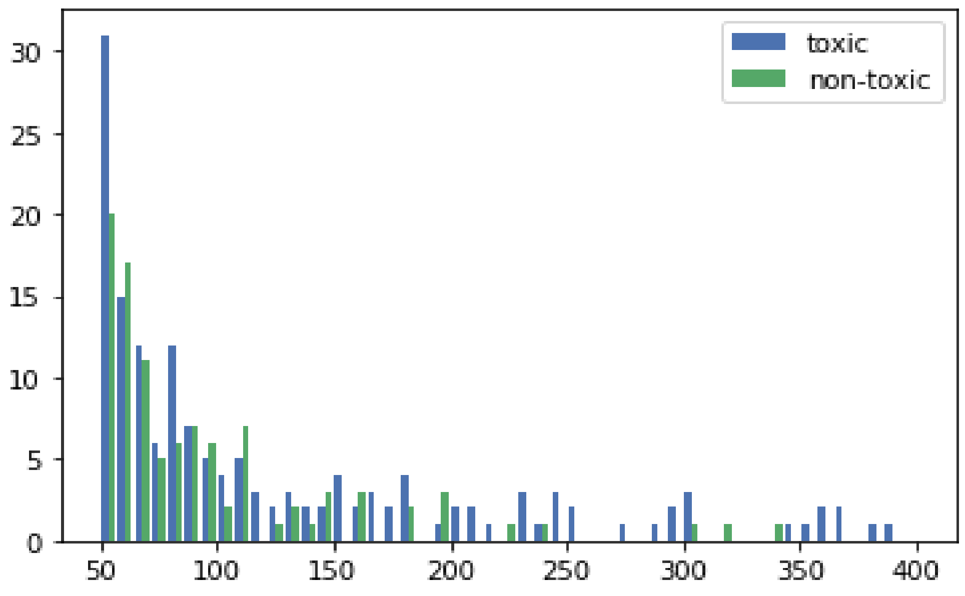


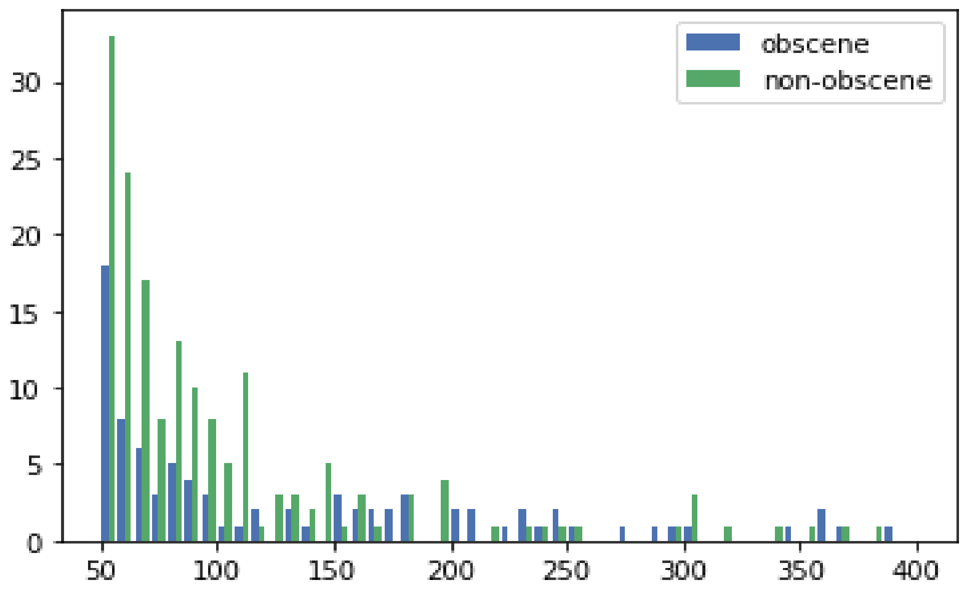
2.2.2 Number of uncommon punctuation



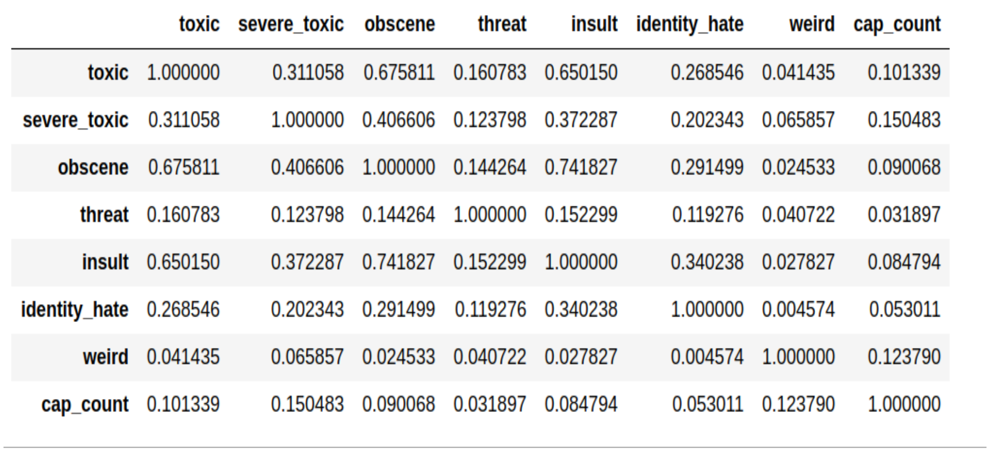


2.2.3 Number of capital words





As we can see in the above pictures, the distribution of ‘clean’ and the distribution of ‘non-clean’ comments follow a similar trend. The increase in magnitude of the above features does not show any noticeable change in the distribution that goes against(not followed by) property followed by ‘clean’ comments.
For example, when ‘length of comment’ increases,the frequency of ‘toxic’ and ‘non-toxic’ decreases. Therefore,it is not a distinguishing factor for this classification. So the above features do not yield any extra information for the classification. This is also demonstrated by the correlation matrix given below.



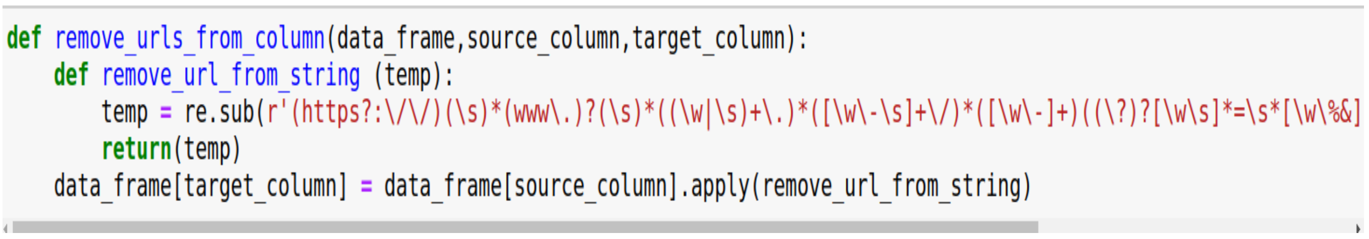
	toxic	severe_toxic	obscene	threat	insult	identity_hate	weird	cap_count
toxic	1.000000	0.311058	0.675811	0.160783	0.650150	0.268546	0.041435	0.101339
severe_toxic	0.311058	1.000000	0.406606	0.123798	0.372287	0.202343	0.065857	0.150483
obscene	0.675811	0.406606	1.000000	0.144264	0.741827	0.291499	0.024533	0.090068
threat	0.160783	0.123798	0.144264	1.000000	0.152299	0.119276	0.040722	0.031897
insult	0.650150	0.372287	0.741827	0.152299	1.000000	0.340238	0.027827	0.084794
identity_hate	0.268546	0.202343	0.291499	0.119276	0.340238	1.000000	0.004574	0.053011
weird	0.041435	0.065857	0.024533	0.040722	0.027827	0.004574	1.000000	0.123790
cap_count	0.101339	0.150483	0.090068	0.031897	0.084794	0.053011	0.123790	1.000000

2.3 Pre-processing

Before the correct model is built we should convert these strings to a numerical form which is understood by the model. Before we convert strings to numerical form we will pre-process the strings to remove unnecessary information from them. Following are some techniques we are using to pre-process the given **strings**.

2.3.1 Remove URLs and links from the strings

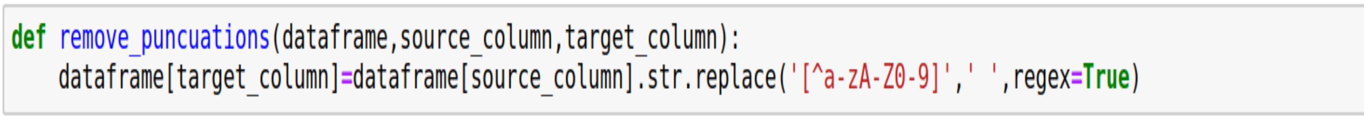
Few strings include links to other sites and URLs which are not useful features in deciding whether a feature is useful or not,so the urls are dropped from comments.



```
def remove_urls_from_column(data_frame,source_column,target_column):
    def remove_url_from_string (temp):
        temp = re.sub(r'(https?:\/\/)(\s)*(www\.)?(\s)*((\w|\s)+\.)*([\w\-\s]+\/)*([\w\-]+)((\?)?[\w\s]*=\s*[\w\%&]
        return(temp)
    data_frame[target_column] = data_frame[source_column].apply(remove_url_from_string)
```

2.3.2 Dropping numbers and punctuation

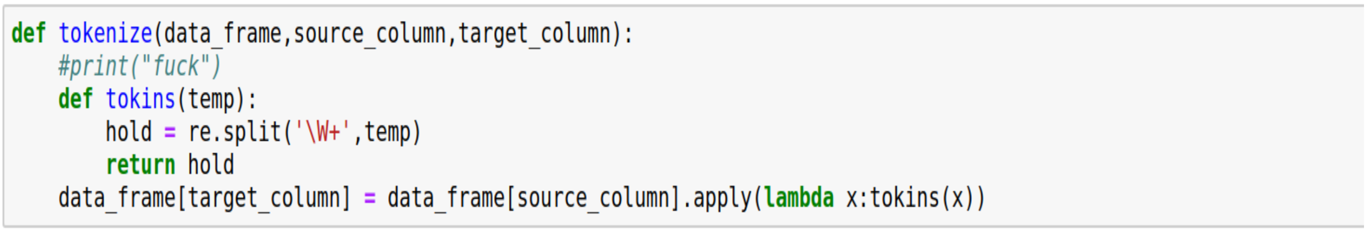
Numbers and punctuation are dropped from the comments as they contribute nothing to classifying the comments. Uncommon symbols are computed before punctuation are dropped. The output after applying this on a comment is a single string which is converted into lowercase alphabet.



```
def remove_puncuations(dataframe,source_column,target_column):
    dataframe[target_column]=dataframe[source_column].str.replace('[^a-zA-Z0-9]',' ',regex=True)
```

2.3.3 Tokenization

The string from above is tokenized so that every comment is broken down into an array of words that make up the comment.



```
def tokenize(data_frame,source_column,target_column):
    #print("fuck")
    def tokins(temp):
        hold = re.split('\W+',temp)
        return hold
    data_frame[target_column] = data_frame[source_column].apply(lambda x:tokins(x))
```

2.3.4 Removing stopwords

These are frequently used words (ex:can,have,thus) that have no effect on the classification of our comments. They are removed from the tokenized array by using **stopwords** library.

```
def remove_stop_words(data_frame,source_column,target_column):
    stop_words = get_stop_words('english')
    stop_words.append('')
    for x in range(ord('b'),ord('z')+1):
        stop_words.append(chr(x))

    def remove_stopwords(list):
        text = [word for word in list if word not in stop_words]
        return text
    data_frame[target_column] = data_frame[source_column].apply(lambda x: remove_stopwords(x))
```

2.3.5 Lemmatizing and stemmatizeing

First the words are lemmatized (identifies the meaning of the word in a sentence) which gives a proper word ,followed by stemming where they are reduced to root form. Here the words obtained may not be proper words (ex:excitement-¿excit).This greatly reduces the total number of words across the comments. **'nltk'** library was used to do this

```
def lemmatize_and_stemmatize(data_frame,source_column,target_column):
    lemmatizer = WordNetLemmatizer()
    stemmer = PorterStemmer()
    nltk.download('wordnet')
    def lemmatize(temp):
        buff = []
        for word in temp:
            if(word != ""):
                buff.append(lemmatizer.lemmatize(word,pos="v"))
        return buff
    data_frame[source_column] = data_frame[source_column].apply(lambda x: lemmatize(x))
    def stemmatize(temp):
        buff = []
        for word in temp:
            if(word != ""):
                buff.append(stemmer.stem(word))
        return buff
    l = []
    data_frame[target_column] = np.nan
    for i in range (0,data.shape[0]):
        try:
            data_frame[target_column][i] = stemmatize(data_frame[source_column][i])
        except:
            l.append(1)
```

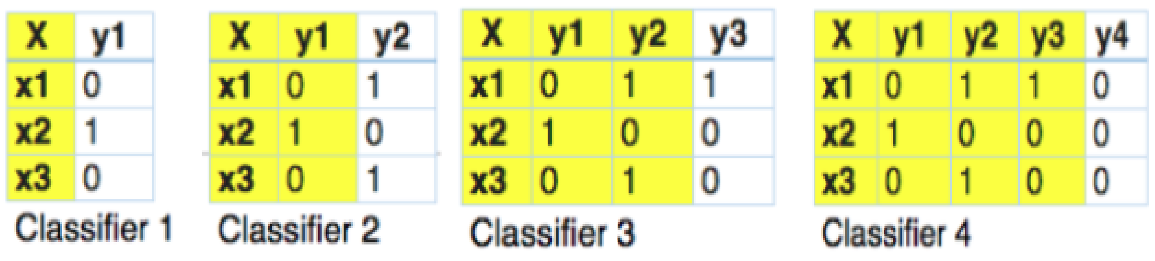
2.3.6 TF-IDF

Now we have obtained an array of words for every comment.These words are joined and a string is constructed again on which TF-IDF is called.This converts our comment which is in string form to numerical form which can be fed to appropriate model along with other obtained features.

```
vectorizer = TfidfVectorizer(strip_accents='unicode', analyzer='word', ngram_range=(1,1), min_df = 75)
vectorizer.fit(data["joined"])
vector=vectorizer.transform(data["joined"])
#victor=pd.DataFrame({'vector':vector})
print(len(vectorizer.vocabulary_))
```

3 Model Selection

We have used Multi-Label Classification Method:



X	y1
x1	0
x2	1
x3	0

Classifier 1

X	y1	y2
x1	0	1
x2	1	0
x3	0	1

Classifier 2

X	y1	y2	y3
x1	0	1	1
x2	1	0	0
x3	0	1	0

Classifier 3

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

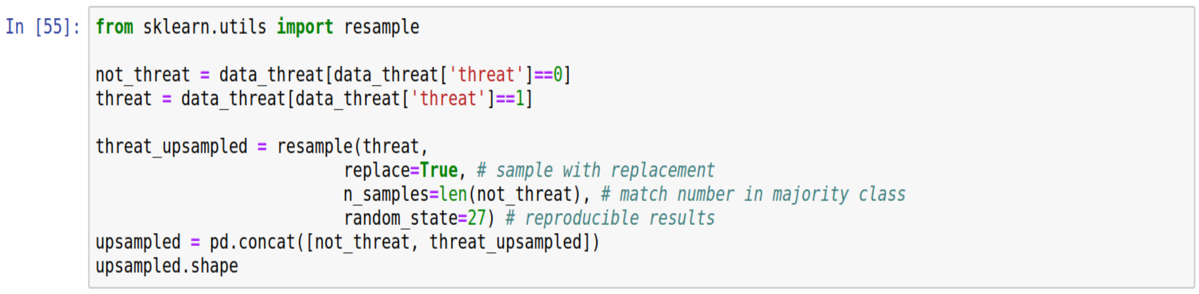
Classifier 4

Classifier Chain Method: We can see that correlation between obscene and insult,obscene and toxic is high from the correlation graph.Also,the fraction of comments classified as 'toxic','insult' from the set of 'obscene' comments is high.From this , we can conclude that classification of whether a comment is 'obscene' or not is a good feature for building models for 'toxic' comments and 'insult' comments.

```
In [52]: temp = data[data["obscene"]==1]
         tox = temp[temp["toxic"]==1].count()[0]
         ins = temp[temp["insult"]==1].count()[0]
         tem=temp.count()[0]
         print("fraction of toxic from obcene")
         print(tox/tem)
         print("fraction of insult from obcene")
         print(ins/tem)

         fraction of toxic from obcene
         0.9378185524974516
         fraction of insult from obcene
         0.7320761128100578
```

Before using a model, we need to observe that the the number of 'identity-hate' and "threat" labelled comments are very low. To train model to accurately identify these types of comments. We need to heavily penalize false negatives(true output =1 , predicted output =0) or we need to over sample data points which are "identity-hate" or "threat". Since these comments are under represented, oversampling of "identity-hate" and "threat" comments is done such tat the data-set created is balenced.

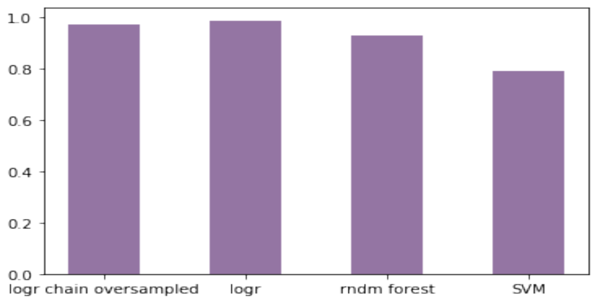


```
In [55]: from sklearn.utils import resample

         not_threat = data_threat[data_threat['threat']==0]
         threat = data_threat[data_threat['threat']==1]

         threat_upsampled = resample(threat,
                                     replace=True, # sample with replacement
                                     n_samples=len(not_threat), # match number in majority class
                                     random_state=27) # reproducible results
         upsampled = pd.concat([not_threat, threat_upsampled])
         upsampled.shape
```

The tag we have are binary. Logistic regression can be used for binary classification across each of the labels.



4 Accuracy

0.97253