TOXIC COMMENTS CLASSIFICATION USING MACHINE LEARNING

A Project Report

For CS 271- Topics in Machine Learning

Presented to

Dr. Mark Stamp

Department of Computer Science

San José State University

By

Vedashree Bhandare [012416924]

Rajeshwari Chandratre [013710242]

April 30, 2019

Table of Contents

I. INTRODUCTION

Maintaining a decorum on an online forum to be able to express and discuss things freely is the need of the hour. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Thus, the task of identifying and removing toxic communication from public forums is critical and is infeasible for human moderators.

Toxic comment classification has become research topic with many approaches proposed recently. In this project, we are tackling the natural language processing (NLP) problem, given the comments, detect different types of toxicity like threats, obscenity, insults, and identity-based hate.

II. DATA SET EXPLORATION AND PREPROCESSING

A. Data Collection

Provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior, the dataset contains three files: train.csv, test.csv and sample_submission.csv. Training data contains comments with their binary labels. Different categories of toxicity are – toxic, severe toxic, obscene, threat, insult, identity hate.

The dataset has about 100k comments and each comment having more than 5 words on an average. Thus, the number of words present were really huge and hence preprocessing was needed to reduce the number of words. Many words occur commonly in English language such as ‘and’, ‘the’, ‘to’, ‘it’, ‘of’, etc. Such frequently occurring words are referred to as stop words and do not really contribute to the toxicity of the comments. We also need to bring different forms of words in uniform format. E.g., the forms such as ‘runs’, ‘running’, ‘ran’ should be converted to ‘to run’. Hence, the comments containing these word formats need to undergo some preprocessing which we will see further. We performed following steps as part of preprocessing.

B. Data Analysis

Data Analysis is one of the most important aspects when it comes to building a Machine Learning model. Unless we know what kind of data we are dealing with, we can not make informed decisions further in our modelling process. For our project, the data size was pretty huge. The precise number of records in our training data set was 159571, whereas the number of records in our test dataset was 153164.

The training dataset has 8 columns, a unique identifier ‘id’ for each comment, the comment in the form of textual data in the column ‘comment_text’, and the 6 categories into which we need to classify our data, namely ‘toxic’, ‘severe_toxic’, ‘obscene’, ‘threat’, ‘insult’, ‘identity_hate’. Each category contains the labels in terms of ‘0’ or ‘1’.

The test data set has only two columns namely id, and comment_text. It does not have the categories in which the comments are classified meaning that it is an unsupervised problem. Since there is a certain difficulty associated when dealing with unsupervised dataset classification and then to compute of accuracy level of this unsupervised test data, we are using some fraction of available training data as our test data. We achieve this by using sklearn’s train_test_split. Thus, our testing will be performed on the test set which is obtained by splitting our training data.

Now, let’s get to visualizing the data. To study the distribution of data belonging to each category, we plot a bar graph which gives us clear understanding of number of comments present in each category.

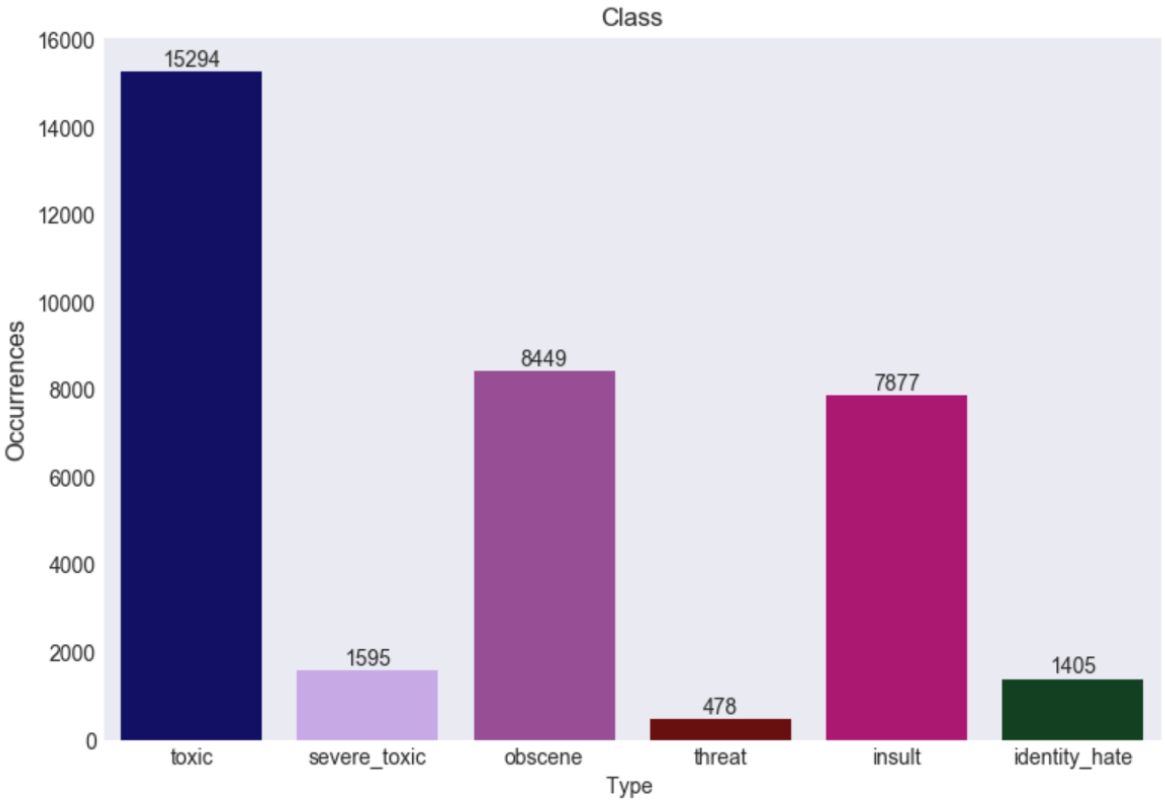

Figure 1: Distribution of number of comments across different types of categories

From the Figure 1, it is evident that the data is imbalanced. The data is not equally distributed. Amongst the toxic categories, the 'threat' category has the minimum number of comments while the 'toxic' category has the maximum number of comments.

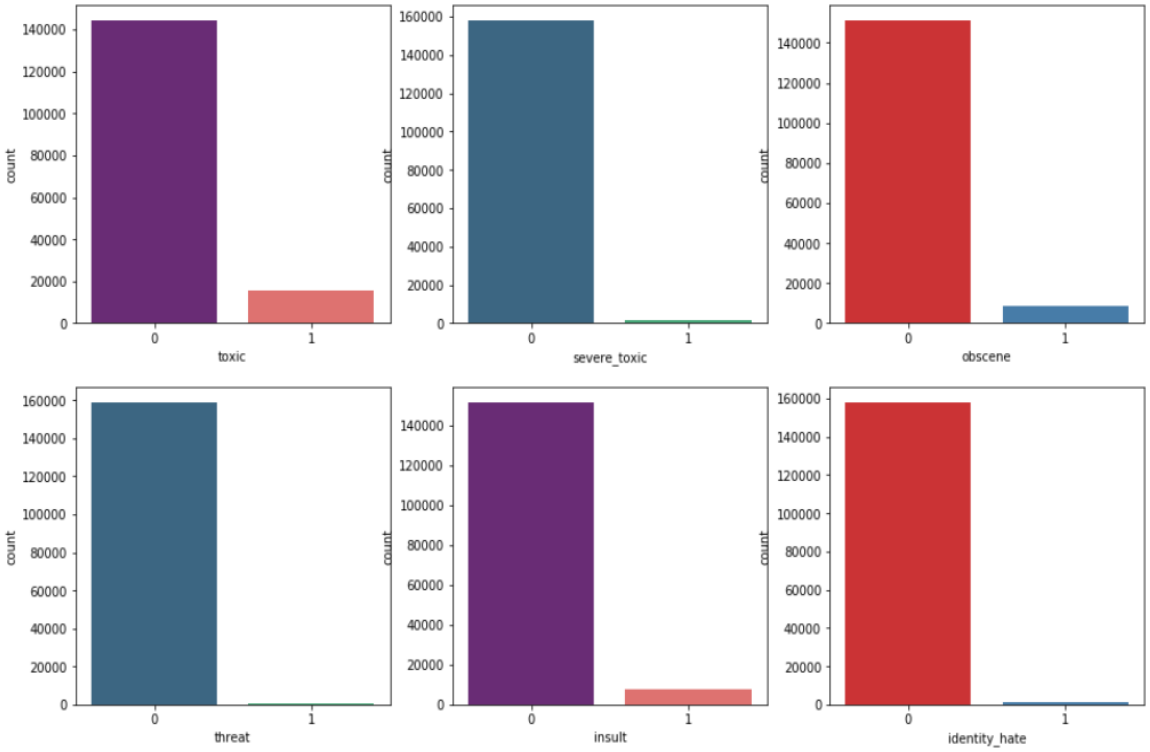

Figure 2: Distribution of each category with respect to the remaining categories

We are also visualizing distribution of data belonging to each category with respect to the records in all the other categories put together. We get the plot described in Figure 2 which summarizes what is the number of comments in each category as compared to the number of comments in all the rest categories put together including the ‘clean’ or ‘non-toxic’ data.

Figure 3: Word Cloud for the dataset containing toxic comments

Figure 3 shows the words most represented in our toxic comments' dataset.Word clouds are a visually appealing way to see which words are dominant in a text. With the help of word clouds, one can quickly and easily understand the vocabulary in the textual dataset.

C. Data Preprocessing

Various NLP techniques can be applied to the text data in order to make it fit for further text analysis. Let us look at the techniques we used to preprocess our raw text data.

a. Removing Stop words

In the first step, we removed all the stop words and punctuations from each comment and every word was converted to lower case letters. The removed stop words are the commonly appearing words in English language such as ‘the’, ‘and, ‘of’, ‘it’, ‘to’, etc.

b. Stemming

In the next step, we performed stemming, process that chops off the ends of words by removing the derivational affixes. PorterStemmer from nltk was used for this purpose.

c. Lemmatizing

Lemmatization is the algorithmic process of determining the lemma of a word. We group the inflected form of the words together to analyse them. Like “to walk” can be expressed as walking, walked, walks. So, we convert all these forms to a lemma i.e walk. NLTK library’s WordNetLemmatizer was used for this task.

III. FEATURE ENGINEERING

A. Vectorization

Machine Learning models only work with numerical data, so we needed a mechanism to convert the textual data to numerical data. In our initial approach, we tried working with Term Frequency Inverse Document Frequency (TF-IDF). TF-IDF assigns how important a word is in all the documents that were given as an input. If a word occurs many times a document, it is considered important. But, if the same word occurs in multiple files, it is assigning less weightage as it could be one of the common words.

TF-IDF has a problem that it does not consider the sequence of the words in the text and is only based on the frequency of the word. This is bad because the sequence of word does matter for finding the importance of the words. For e.g. the sentences "*This food is good, not bad at all"* and *"This food is bad, not good at all"* have the same TF-IDF but the meaning of the two words is completely different.

We used sklearn TF-IDF Vectorizer with word analyzer and n-grams set to 6 words. The results obtained from the TF-IDF were not good and hence another approach was required.[1]

B. Word Embeddings

Another approach for converting data to vector forms is to use word embeddings. We used GloVe for the same. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. It combines the advantages of the two major model families in the literature: global matrix factorization and local context window methods.

Resulting representations showcase interesting linear substructures of the word vector space. Main intuition underlying the model is the simple observation that ratios of word-

word co-occurrence probabilities have the potential for encoding some form of meaning. Since huge amount of data is required by the model to learn the co-occurrence probabilities we used a pre-trained model which is trained on 1 billion words with vocabulary of 400 thousand words.

GloVe has embedding vector sizes: 50, 100, 200 and 300 dimensions. We chose the 100-dimensional one and set the trainable option of the LSTM model to “False”. This was done using embedding_index dictionary which stores the word vectors for a given word.[2]

IV. IMPLEMENTATION

As part of implementation we applied various machine learning models like KNN, Multinomial Naive-Bayes, SVM, XGBoost and Neural Network. We applied these models using vectors obtained using TF-IDF and Co-occurrence matrix obtained using pretrained GloVe Models.

A. Algorithmic techniques to build a multi-label classifier

There are multiple techniques to build a multi-label classifier such as Binary Classifier, Classifier Chaining, and Label Powerset.[3] Let us see deeper into the techniques which have been used in this project. In Binary Relevance technique, each label is treated as a separate class. It amounts to independently training one binary classifier for each label[3]. Without any regards to other labels, a single classifier deals with a single label. In our problem, comments are the independent variables and the types of toxicity are its target variables. In binary relevance, this problem is broken down further, where each toxic type (e.g. severe_toxic, obscene, insult, etc.) will be considered each as a separate classification problem and operated further in a way how a normal classification problem would be computed. This approach is simple and efficient; the only drawback being it cannot consider any correlation between the labels if present, as it treats every target variable independently[3]. The other approach that we tried for this multi-label problem was Classifier Chaining. In this technique, the first label/classifier is trained just on the input data, while all the other labels/classifiers are trained on the input data as well as previous classifiers in the chain. This approach helps to overcomes the drawback in the Binary Relevance technique. It maintains the correlation between labels. Classifier Chaining is otherwise similar to Binary Relevance; the only difference being the former preserves the

label correlation by forming classifier chains[3]. Thus, we tried to build our models using these two techniques used for computing multi-label classification problem.

B. Implementing Machine Learning Model + TF-IDF vectors-

SVM: With the help of Support Vector Machines, we can classify the data into two groups by passing a linearly separable hyperplane through a dataset. This hyperplane is a linear separator that can be applied to for any dimension; A hyperplane could be a line, plane, or a hyperplane in two, three, and four dimensions respectively. This is one of the most effective techniques for classification problems. But since our problem being multi-label classification, we make use of another variant of SVM called SVC, which is suitable model to be used in case we have multiple classes to categorize our data. The Support Vector Clustering model (SVC) organizes the data into clusters according to some criterion in an attempt to organize data into a more meaningful form. In SVC data points are mapped from data space to a high dimensional feature space using a kernel function[4]. In our implementation, we have used sklearn's SVC package with the linear kernel and class_weight parameter set to 'balanced'.

KNN: K-Nearest Neighbors is the easiest Machine Learning algorithm. Its purpose is to use a dataset in which the data points are separated into multiple classes. It helps in predicting the class of a new data point belonging to such dataset. It is also said to be a lazy algorithm, meaning that it does not use the training data points to do any generalization. Here, the training phase is minimal meaning that the training phase is very fast. This lack of generalization and minimal training phase leads to the KNN needing all the training data in its testing phase[5]. Thus, KNN becomes a lazy learner as it uses the entire training data

in its testing phase. A KNN decides class membership based on majority votes of its neighbors, a data point is classified into a class which is most common to its K nearest neighbors. This algorithm is computationally expensive as it stores all the training data and comparatively is much slower than most other ML techniques. We set the 'k' or the n_neigherest_neighbors parameter equal to 10. We also tried it with k = 6 since we have 6 different categories.

Multinomial Naive Bayes: It is a sub-type of Naive Bayes algorithm. Multinomial Naive Bayes is a technique that is typically used in document classification where occurrence of a word in a single document is represented by events. In this method, our samples are represented by feature vectors. They are indeed the frequencies with which certain events have been generated. It estimates the conditional probability of a particular word given a class as the relative frequency of term t in documents belonging to class c. The variation takes into account the number of occurrences of term t in training documents from class c, including multiple occurrences[6]. In our implementation, we have used scikit-learn's MultinomialNB classifier with values for the smoothing parameter α set to 1 and fit_prior set to true.

XGBoost: XGBoost stands for eXtreme Gradient Boosting. It is optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. The design goal of this algorithm is the efficiency of compute time and memory resources. XGBoost makes the best possible use of available resources, supports the parallelization of tree construction, and ensures continued training meaning it can further boost an already fitted

model on new data.[7] In our implementation, the scale_pos_weight which is a parameter that controls the balance for positive and negative weights is set to 1. Other parameters that we experimented were number of threads set to maximum threading, and depth of the tree set to 4.

C. Implementation of LSTM + Glove

LSTMs are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well with time series data, avoid the long-term dependency problem and vanishing gradient problem that traditional RNN face.

The key to LSTMs is the individual cell states in the model have the ability to remove or add information to the cell state through regulated structures called gates layers. This is useful in practice because it allows the model to remember insights derived from words throughout the sentence.

First layer consists of a sigmoid layer called “forget gate layer” which is used to decide what information is to be stored and what needs to be thrown away. In the next step it is decided what new information is stored in the cell. It uses 2 functions sigmoid to decide what value to update and tanh to create an output vector. Finally, we run a sigmoid layer which decides what part of cell’s output are we going to output and then multiply it with tanh function to push output between 1 and -1. [8]

For our training purpose, we used 40k comments from the train set and validation was done on 2k comments. In our project we used Keras implementation to build and compile LSTM model. It consists of an embedding layer, one densely connected layer with 128 units across the concatenated word vectors for each of the words in the comment obtained using GloVe

followed by a dropout of 10%. The model finally consists of an output layer with a sigmoid activation that makes the prediction for the multi-label classification.

We used binary cross entropy loss which handles each toxicity class independently instead of as a 6-dimensional vector[9].

V. RESULTS

Let us look at the results we got by implementing different Machine Learning models

A. Accuracies of different Machine Learning Models

Machine Learning Models	Accuracy
LSTM	96.3700
XGBoost	91.3444
KNN	89.9722
SVM	89.6944
MultinomialNB	89.0111

Table 1: Accuracies of different Machine Learning models

We can see that the accuracies of different Machine Learning models were quite impressive, suggesting that the classifiers are performing well on distinguishing comments. There is also an observation that all the ML techniques other than LSTM exhibit accuracies which are more or less in the same range, but only LSTM outperforms this trend by achieving an accuracy as high as 96%. We have explained the reason behind this later. We observe a similar trend in ROC-AUC values as well. ROC curve uses True positive Rate and False Positive Rate to evaluate the model performance. ROC plots for each of the classifier below gives us the real insight about how well the classifiers performed. The obtained ROC curves for different models helped us understand the performance of our models still better as it helped us to know that how many of the actual toxic comments were actually being actually classified in the respective toxic categories.

B. ROC plots of different Machine Learning Models

- **ROC Plot for SVM**

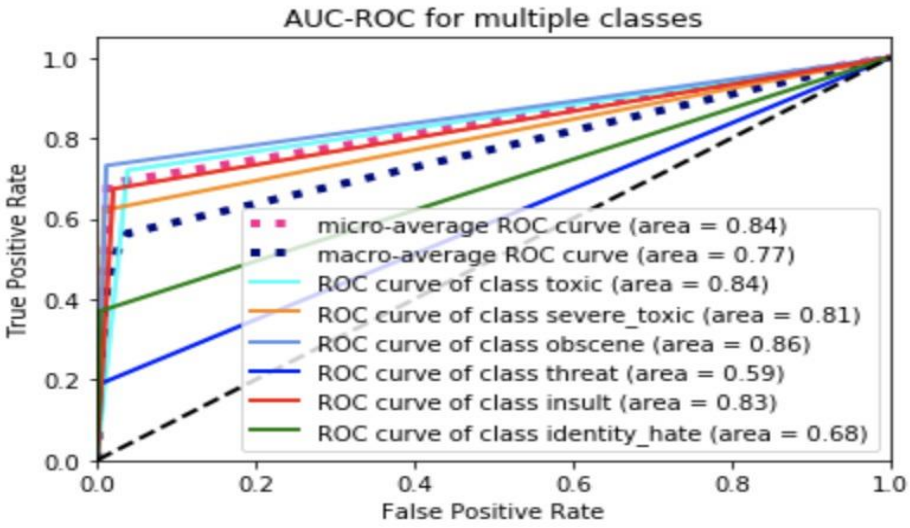

Figure 4: ROC plot for SVM

- **ROC Plot for KNN**

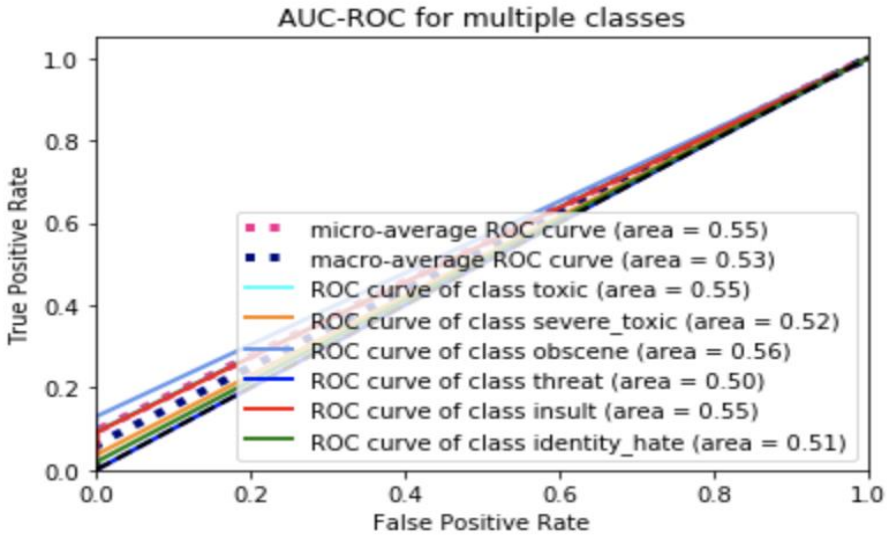

Figure 5: ROC plot for KNN

- **ROC Plot for MultinomialNB**

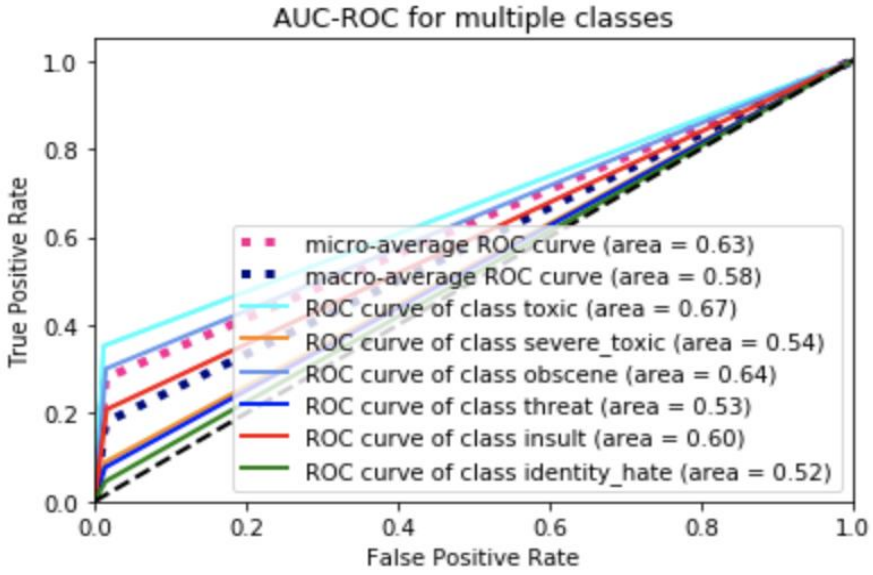

Figure 6: ROC plot for Multinomial NB

- **ROC Plot of XGBoost**

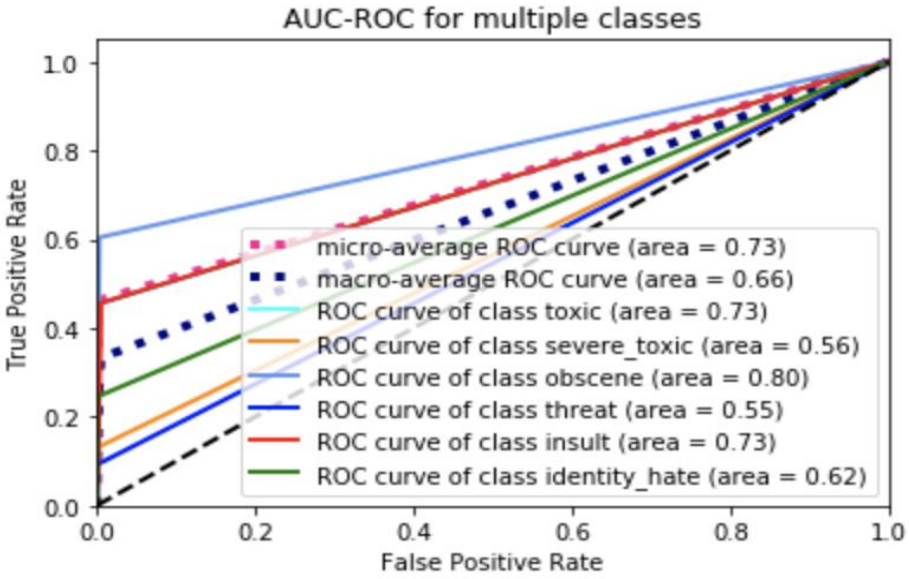

Figure 7: ROC plot for XGBoost

ROC plot basically has the true positive rate along the Y axis and false positive rate along X axis and it evaluates the output quality of the classifier. Since our problem is a multi-label problem, we have drawn ROC curve for each label along with ROC curve by micro-averaging it across all the labels.

Above results represent the ability of each model to distinguish comments into different toxicity level. We can see that the classifiers did not do good job at differentiating comments even though they had very good accuracy.

C. Results for LSTM

With the pretrained GloVe embeddings and LSTM, we were able to get better accuracy of 96% than the previous classifiers, and the plot of the micro averaged ROC and ROC for individual classes is also much better than the previous attempts. For e.g. the micro averaged AUC ROC curve of MultinomialNB is 63.0 while that of LSTM is 88.00. So, we can say that the LSTM performed better than the earlier models. Reason for better results of LSTM are the GloVe word embeddings that were used as the input to LSTM which helped the model learn better than the models that were trained on TF-IDF vectors. Co-occurrence matrix obtained using GloVe captures the information about the context in which the words appeared previously. Helping LSTM to learn and classify word in a better sense. Below is the result of ROC curve for LSTM.

- **ROC Plot for LSTM**

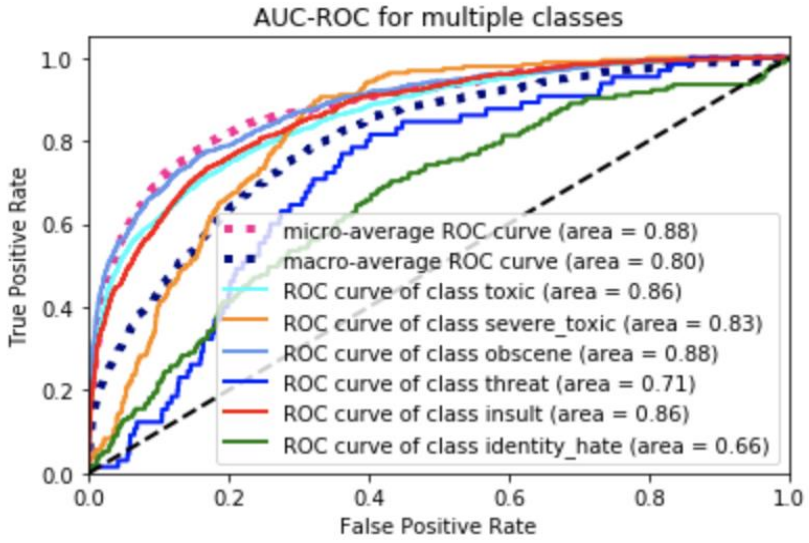

Figure 8: ROC plot for LSTM

D. Comparison of different models:

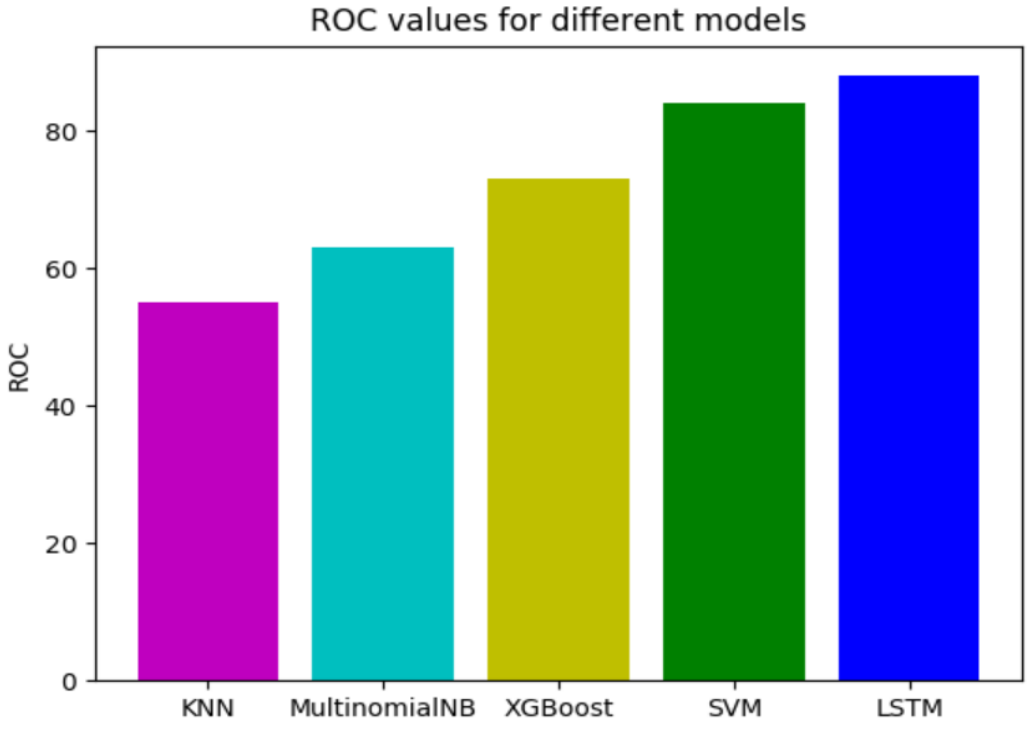

Figure 9: Comparison of ROC values of different models

Model	ROC value
KNN	55.00
Multinomial Naive Bayes	63.00
XGBoost	73.00
SVM	84.00
LSTM+GloVe	88.00

Table 2: ROC values for different models

From Table 2, we observe that different ML models perform poorly to averagely when we compute ROC value but it can be seen that LSTM implementation performs very well, and we are able to achieve a comparatively better ROC value and a good accuracy with this technique. We can see that amongst the different simple Machine Learning techniques that we used, SVM classifier performs better than the rest of them suggesting that it can handle the imbalance in the data well.

VI. CHALLENGES

A. Data Size

The size of the dataset was one of the challenges that we faced. The dataset was about 1.5 GB in size and needed to be loaded alongside the GloVe word embeddings. As the experiments were performed on a commodity hardware with 8 GB of RAM and i7 processor, the hardware was not adequate for the problem. Thus, the experiments that we could perform were limited.

Another challenge was that the test data was unsupervised, meaning that the comments were not labelled as per their categories. Since there was a certain difficulty level associated with the process of unsupervised dataset recognition and then to compute of accuracy level of this unsupervised test data, we decided to use some fraction of available training data as our training dataset.

B. Imbalance in Data

We learnt in the data exploration that the dataset is bulky with the large proportion of unlabelled comments. A very large proportion of data attributes to comments which are not classified into any category. This is resulting in us getting inflated accuracies but inferior ROC-AUC for various ML models we used. Undersampling did not help because it caused the data size reduced to a smaller amount which was insufficient for the models to be trained on.

VII. CONCLUSION AND FUTURE WORK

Imbalanced in the data and very few numbers of toxic comments available in the training set on which model can be trained affected the performance of the model heavily. Though the performance of LSTM is affected by the imbalance, it provides decent results in terms of the AUC under ROC as it is able to learn better with the help of word embedding obtained from GloVe model which captures the context of the words. In future, we plan to understand and explore cost sensitive LST and how will it perform on this dataset. [10]

We also plan to compare the model performance based on embeddings obtained using Word2Vec with the result of GloVe embeddings.

VIII. REFERENCES

[1]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[2] https://nlp.stanford.edu/pubs/glove.pdf

[3]https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/

[4] http://www.scholarpedia.org/article/Support_vector_clustering

[5]https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

[6] https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Multinomial_naive_Bayes

[7]https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[8] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[9] https://keras.io/models/model/

[10]https://www.researchgate.net/publication/321165269_A_LSTM_based_Framework_for_Handling_Multiclass_Imbalance_in_DGA_Botnet_Detection