Toxic Comment Classification

Amay Kadre
akadre | 40909550
akadre@uci.edu

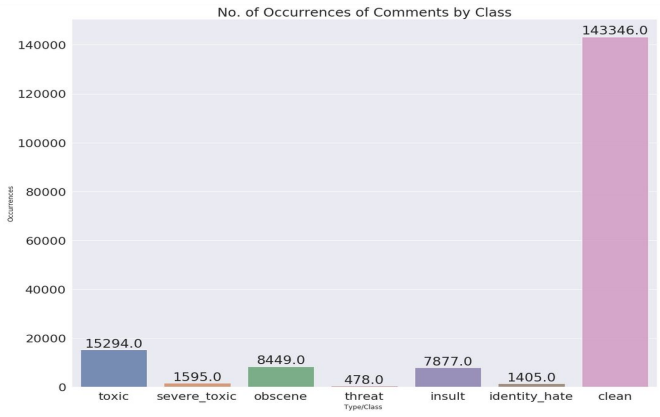
Siddhant Deshmukh
sdeshmu1 | 32887666
sdeshmu1@uci.edu

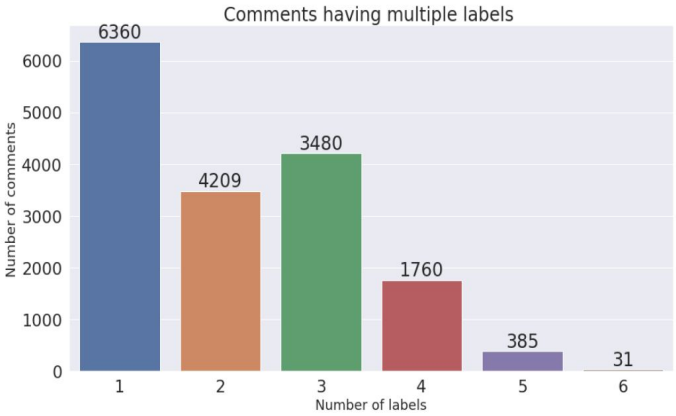
Introduction:

The aim of this report is to provide an approach on identification and classification of online toxic comments and address the famous Kaggle challenge on “Toxic Comment Classification”. There is always a possibility of threats and abuse during online discussions due to which platforms, communities and discussion boards often struggle to provide an effective medium to facilitate conversations. We attempt to solve this problem by implementing Logistic Regression and other Neural Network architectures like LSTM as well as pretrained transformer networks like BERT under the Classification Accuracy metric.

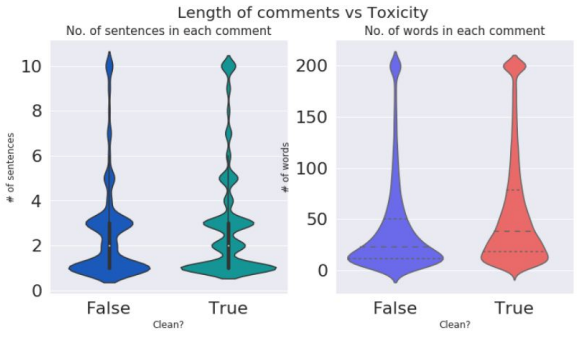
Data Exploration:

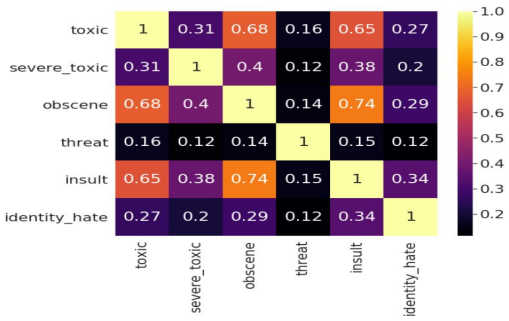
To implement and decide upon various design choices which we used for this project, data exploration served as a significant step. To get a better understanding of the input data, we plotted the following graphs which classify the number of comments for each class:



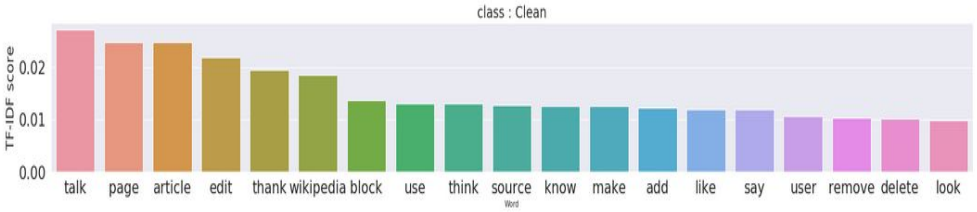


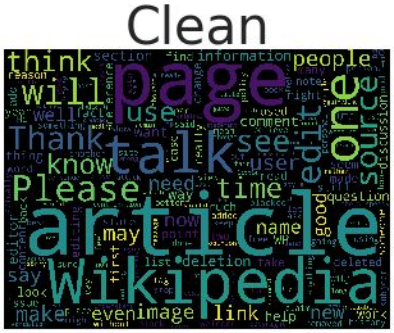
As can be observed, there is an imbalance in categories and we may face class imbalance problems. This is because there are comments with multiple labels and approximately 90% of the comments are observed to be clean. On further analysis of data points using a violin plot, we observe that the toxicity of a comment doesn’t change much when the sentences are very lengthy. Thus, during the data pre-processing step, we implemented a design choice of reducing the length of comments. Furthermore, on plotting the ratio of co-occurrence of multiple labels, as shown below, we come to a conclusion that “toxic” seems to be a superset of all other classes.





One more significant observation was obtained using the TF-IDF Vectorizer, provided by sklearn for feature extraction, over the frequency distribution of words (unigram). The TF-IDF Vectorizer counted words in text corpus and penalized words which are too frequent. Following is the graph (shown for class: “clean”) wherein we observed the words with the top TF-IDF scores for each class. This would serve as a feature for further model implementations.





Data Preprocessing:

- **Removal of some comments’ features:** Some features of a comment do not help much in toxic classification and thus can be removed. Some examples are punctuation marks, non-ASCII special characters, numbers and words with length greater than 25.
- **Stopwords removal:** Frequently used words which do not have a positive or negative impact (‘a’, ‘an’, ‘the’ etc.) and thus no significant value to the comment can be ignored during preprocessing. To perform this preprocessing we used Python’s built-in dictionary of stop words.
- **Stemming:** To achieve the training with a better accuracy, we converted the inflected/derived words into their word stem or root form. This was achieved using the Snowball Stemmer present in nltk.
- **Sentence Padding:** As observed from the violin plot, if the comments are very long i.e. for sentences having words greater than 150, the toxicity isn't affected much. Hence, we trim longer sentences and add padding to the shorter sentences by filling the shortfall with zeroes to make them the same length. We used the max length of sentences to be 150.
- **Implementing TF-IDF Vectorizer:** Words with very high frequency don't provide any valuable insight about a comment. As observed from the TF-IDF score, which penalizes such words, we use the term frequency and the Inverse Document Frequency to disregard such words.

Model exploration:

LSTM (Long Short Term Memory):

Introduction:

As inferred from the data exploration, there is a sequential relationship between the words in a sentence. LSTMs provide great performance for many sequence-related problems. These are specifically designed to "remember" what was said earlier (or even later) in a sentence.

This allows LSTMs to better capture language context, improving its ability to capture semantic meaning - this makes it a very powerful option for text classification where semantics is very crucial. Hence, LSTM became our obvious choice to solve this challenge. Moreover, we used GloVe as an unsupervised learning algorithm for obtaining vector representations for words. The advantage of using GloVe is that unlike other representation schemes such as Word2vec, it incorporates global statistics (word co-occurrence) to obtain word vectors and not just the local statistics or local context information of words.

LSTM Implementation:

Since we can't send the input in the form of words directly to our model, we first need to convert the words obtained from the training data into word embeddings. Firstly, we use standard Keras preprocessing to convert the comments into a list of word indexes of equivalent length. We determine this length based on our data visualization and set it to 150 (refer to the violin plot above) as part of our hyperparameter tuning. We also set another hyperparameter i.e. the maximum no. of features to 10000, based on the occurrences of unique words. We then send our list of encoded sentences into the Input Layer.

We defined an Input Layer which accepts a list of sentences that has a dimension of 150. We also created a dictionary lookup of these encoded words into word embeddings based on the GloVe pretrained vectors. We then created a simple Bidirectional LSTM with a dropout of 10% enabled since even 2 epochs are enough for the model to overfit. Since the output from the previous layer is a 3-D tensor, we convert it into a 2-D tensor using Global Max Pooling. We then connected the previous output to a densely connected layer with the ReLU activation function. Finally, after another dropout layer, we feed it to a sigmoid function. This is done because this is a binary classification problem with 6 labels, and this function will compress the output between 0 and 1 for each label. We use 'Adam' as our optimization function and binary cross entropy loss, since this is a binary classification problem.

ReLU vs SELU:
ReLU (Rectified Linear Unit) is a very popular activation function in the field of Neural Networks. However, it always outputs 0 if the values are negative, and this may cause some neurons in the network to never update to a newer value, thereby rendering a part of the network useless (Dead ReLU Problem). It also faces the exploding gradient problem. To combat this problem, a new form of activation functions called ELU (Exponential Linear Unit) came to light and SELU (Scaled Exponential Linear Unit) is one of the modern ones. It fixes the exploding gradient problem, and we also found out empirically that using SELU resulted in a faster convergence and higher accuracy. Hence, we chose SELU as our activation function in our final model.

Adam vs Nadam:
Adam can be perceived to be a combination of RMSProp and SGD with momentum. It is a fairly popular optimization algorithm which managed to achieve excellent results. More recently, a new optimization algorithm called Nadam was introduced which demonstrated the benefits of Nesterov momentum over normal momentum. Thus in our project, we tried to analyze the effects of using these two optimizers. We found out that Nadam gave better results on training accuracy as compared to Adam. However, Nadam had a slightly reduced performance on the validation accuracy.

Adaptation to Overfitting:
After observing the Validation scores, we realized that the SELU models had a higher accuracy in the first epoch as compared to the second (Possible Overfitting). Hence, when we submitted the model, we trained it only on one epoch. Hence, the model obtained with the above mentioned hyperparameters (SELU + Nadam with 1 training epoch) gave the best possible outcome for LSTM.

Hyper Parameters	Training Accuracy (epoch 1, epoch 2)	Validation Accuracy (epoch 1, epoch 2)
ReLU + Adam	0.9782, 0.9823	0.9814, 0.9825
SELU + Adam	0.9796, 0.9829	0.9819, 0.9822
ReLU + Nadam	0.9801, 0.9829	0.9817, 0.9823
SELU + Nadam	0.9804, 0.9828	**0.9826**, 0.9820

Logistic Regression:

Introduction:
Since we need to classify a comment into the given categories, implementing a logistic regression classifier to disjointedly classify two categories at a time comes out to be a natural choice. Hence, individual classes are predicted using binary classification with logistic regression.

Logistic Regression Implementation:
The main idea is to implement logistic regression for each class and calculate the cross validation score for each class. First, we initialize word and character vectorizers using the TF-IDF Vectorizers present in sklearn feature_extraction module. We set the max features for word vectorizers to be 20000 and for character vectorizers to be 30000. Later, we obtain the training features using the union of the previously obtained word and character vectorizers. Henceforth, for each class, we predict the model probability for a given comment.

We trained the model on "liblinear" solver which is good for one-versus rest schemes. This provided an accuracy of 0.9819756. Then to optimize our cross validation accuracy, we decided to optimize the objective function by using "sag" algorithm for training which is faster and a better choice for large datasets and also suitable for multiclass problems. Although for default max iterations (100) and no penalty the model provided bad accuracy for cross validation, tuning the penalty to L2 and testing various values of maximum iterations led to a better accuracy. As can be observed from the table below, a lower value of max_iters provided better results. The optimal value for our Logistic Regression model was obtained at max_iters = 8 and penalty = L2.

Solver / Algorithm	Max Iterations	Penalty	Cross Validation Accuracy
liblinear	100	L1	0.9819756
sag	100	none	0.9792850
sag	100	L2	0.9819798
sag	8	L2	**0.9819912**
sag	16	L2	0.9819865
sag	32	L2	0.9819843
sag	64	L2	0.9819829
sag	128	L2	0.9819823

Training Performance	Validation Performance	Testing Performance
0.9855	0.9820	0.97934

BERT (Bidirectional Encoder Representations from Transformers):

Introduction:
BERT a multilingual transformer based bidirectional model has achieved state-of-the-art results on various NLP tasks. As it is based on the transformer architecture, it replaces the sequential nature of RNN (LSTM & GRU) with a much faster Attention-based approach.
BERT is pre-trained on masked language modeling and next sentence prediction which allows us to fine-tune the model on downstream specific tasks such as sentiment classification, intent detection, question answering and much more.

In traditional classification based models, each task is assigned to one and only one label which is also called as multi-class classification. BERT, on the other hand, incorporates multi-label text classification which assumes that a task can be simultaneously and independently be assigned to multiple labels. For this project, we have used DistilBERT, which is a small, fast, cheap and light Transformer model trained by distilling Bert base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of Bert's performances as measured on the GLUE language understanding benchmark.

Thanks to transfer learning, we can achieve high performance using pre-trained models like BERT. We reduced the batchsize and gradient accumulation step size to reduce the amount of memory consumption because DistilBERT, although having a lower number of features than BERT, still has a high amount. We also set the max sequence length to 150, based on our EDA.

Comparative Study on Models:

We observe that Logistic Regression gave a very high Training Performance, however on testing it had the lowest score among the three. As expected, DistilBERT gave the best overall performance.
Despite its performance, DistillBERT took a long time to train (approximately 11 hours). While on the other hand, Logistic Regression had the lowest training time (approximately 20 mins) and LSTM provided optimal performance with approximately only 1.5 hours of training.

Model	Training Performance	Validation Performance	Test Performance
Logistic Regression	0.9855	0.9820	0.97934
LSTM	0.9804	0.9826	0.98125
DistilBERT	0.9874	0.9852	**0.98256**

Acknowledgements:

We thank Professor Xiaohui Xie for providing an excellent course on Machine Learning, as well as our TAs Deying Kong and Xiangyi Yan for their assistance with challenges faced and overall guidance.

References:

LSTM. Kaggle (SBongo) - LSTM using Keras
Glove Paper. NLP Stanford Publication: gloVe
SELU Paper. Self Normalizing Neural Networks
DistilBERT.DistilBERT Transformer- documentation