Sandhya Chandramohan - UCI ID - 61167891
Naveena Kannan - UCI ID - 50964330

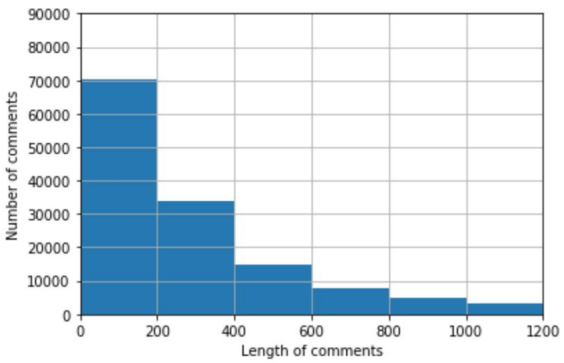
Toxic Comment Challenge

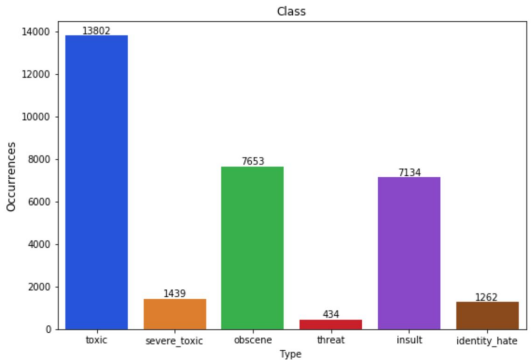
1.1 Problem Description

One of the biggest problems with social media and online forums is the presence of negative online behavior. The so-called ‘trolls’ often post negative comments online that are rude, disrespectful, inflammatory and abusive. The Toxic Comment Challenge aims to build a multi-label classification model that detects different levels of toxicity in online comments. The 6 levels being toxic, severe toxic, obscene, threat, insult and identity hate.

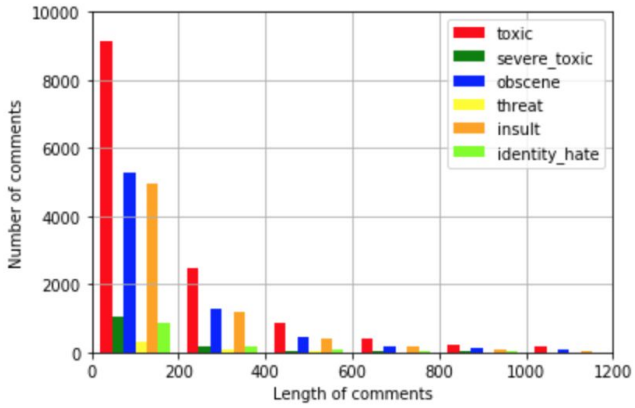
1.2 Data Exploration and Visualization

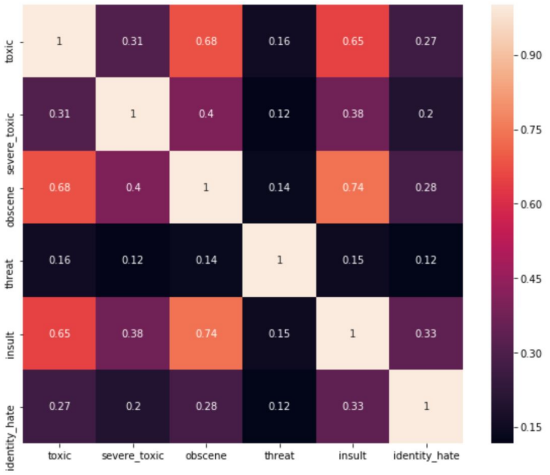
The next step was to explore the dataset and identify various patterns in it. The data_loader module was used to load and split it into training, validation and test datasets. The training dataset is made up of 143645 comments with their labels, the validation dataset with 15926 and the test dataset with 63978 comments.
Exploring the test dataset here on, we counted the frequency of occurrence of multi-labelled data and noted that there were 129006 samples with all-zero labels or non-toxic comments, 14639 samples with at least one non-zero label and only 8941 samples with 2 or more non-zero labels. This shows that our data primarily comprises of non-toxic comments with only about 10% of the comments being toxic. We also note that the comments contain 496117 unique words of which the 10 most common words are "the" "to" "of" "and" "a" "I" "is" "you" "that" "in". One issue here is that we are considering uppercase words as being different from lower case words and including other special characters that aren't really useful for our goal. We will perform a data cleanup in the next step. Visualizing the classification of comments by length of the comment we find that the average comment length prior to processing the data is 393.746. Plotting the classification of the comments according to the toxicity level they belong to we find that most of the toxic comments are labelled “toxic” (13802), with comments labelled “obscene” and “insult” coming in next at about 7000 each. We next try to find any correlation between the length of the toxic comment and it’s label, by grouping the plot of comments by length by the class they belong to. We find that most of the toxic comments over 90% of them are less than 400 characters in length. This observation enables us to make decisions for pre-processing and preparing the data to train, in the next step.





We also generate a heatmap correlation matrix to observe any correlation between the features. Consistent with our previous observations we find that the “toxic”, “obscene” and “insult” features have a higher correlation than the rest.





1.3 Data Pre-processing

As part of the data pre-processing step, we take various steps which include splitting each comment by whitespace, removing the punctuations, changing all letters to lowercase, removing any characters which are not alphanumeric. We then utilize the ‘stopwords’ corpus from the nltk library to remove any stopwords. Stopwords are the common words in the English language which do not have any significant impact on the meaning of the sentence. Following this we lemmatize the words. Lemmatizing is the act of grouping together inflected forms of a word so that they can be analysed together. For example: “lemmatize”, “lemmatizing”, “lemmatized” etc. Lemmatizing also takes into account the context and meaning in which the word is used. For this we use the ‘wordnet’ lexical database from the nltk library. Now, this gives us clean data for use in the various models we use to analyze our performance.

1.4 Techniques and Models

We use Multinomial Naive Bayes and Logistic Regression as our baseline models, post which we try to improve our performance using various complex models. The 3 other models on which we evaluate performance are Recurrent Neural Networks with LSTM (Long Short Term Memory) architecture, Convolutional Neural Networks and LightGBM (Gradient Boosting).

As the first step in our process, we convert our pre-processed clean data into a word-embedding for usage in our models. We use the word2vec model which is a numerical representation of contextual similarities between words to generate word vectors for both the neural networks, CNNs and RNNs. Word2Vec is a state-of-the-art algorithm for word similarities/analogies that internally combines, the two techniques of CBOW (or Continuous Bag

of Words) and the Skip-gram model. We use pre-trained word2vec models for our purposes. But since word2vec features sometimes contains negative values which is not compatible with Naive Bayes or the LightGBM models, we use a frequency based word-embedding method called TF-IDF vectorization in both. TF-IDF vectorization computes the ‘relative frequency’ of a word that it appears in a document, compared to its frequency across all documents. Following this we are now ready to form and train our models.

1.4.1 Naive Bayes - Logistic Regression (Baseline)

We use the Logistic Regression (Naive Bayes Model) as our baseline model. We also compare the performance of this model against vanilla Multinomial Naive Bayes and Logistic Regression models. The Logistic Regression (Naive Bayes Model), we use the Naive Bayes feature equation and fit the model one dependent at a time. This model closely approximates the SVM model. The technique and process used here is straightforward and involves no hyperparameter tuning. Naive Bayes assumes that the features are independent and calculates posterior probabilities, whereas Logistic Regression optimizes the log-likelihood function with probabilities modelled by the sigmoid function. Both these approaches are not sophisticated enough to achieve good accuracy. Although the performance of the models as evaluated under the ROC-AUC curve is not great, it provides a baseline model to compare against.

1.4.2 RNN (LSTM)

Recurrent Neural Networks (RNNs) are most commonly used for problems in natural language processing. They exhibit temporal behavior and capture sequential data which is beneficial when dealing with textual data. We use the LSTM (Long Short Term Memory) architecture in our RNN model which contains a gated cell which acts as a memory unit. In our model we used the pre-trained FastText embeddings from Facebook to produce a 300 dimension word vector of each word, post which our vocabulary size was 287514. Our model consists of first an embedding layer, followed by an LSTM layer with 60 cells, followed by max pooling layers and fully connected layers at the end. The first dense fully connected layer has a ‘relu’ activation followed by a dropout layer and another fully connected dense layer with ‘sigmoid’ activation. We use binary cross entropy as our loss function. We varied the values of dropout and observed that the best performance is observed for a value of 0.3.

1.4.3 GLoVe + RNN

Transfer Learning is a machine learning technique where a model trained on one task is repurposed on a second related task. It is common to perform transfer learning with natural language processing problems. An embedded matrix is created using Stanford's GloVe pretrained Model, followed by a vanilla RNN layer. Relu and sigmoid activation is used in the first and second dense layer. We have set the dropout rate as 0.25 as it gives the best performance. There is not much variance in the performance by tuning other parameters. Hence to improve the performance of the classifier more focus was put on model architecture than

parameter tuning. The word embedding layer was training with a better initializer, hence this model gives the best performance.

1.4.4 CNN

CNN is used here for sentence classification. A standard model for document classification is to use an Embedding layer as input, followed by a one dimensional convolutional neural network, pooling layer, and then a prediction output layer. Kernel size in the convolutional layer defines the number of words to consider as the convolution is passed across the input text document. A basic CNN model with convolution layer, dropout layer and 2 dense layer is tested. By changing parameters, the dropout rate is set to 0.25 as it gives the best performance. The performance does not vary much when tuning the other parameters like size of kernel, size of filters and number of nodes in the dense layer.

1.4.5 LBGM

Light GBM is a gradient boosting framework that uses a tree based learning algorithm. It grows leaf-wise as opposed to depth-wise and performs really well on large datasets. A low learning rate of 0.2 is chosen, along with a bagging fraction of 0.8 specifying the fraction of data to be used for each iteration and a feature fraction of 0.6 meaning 60% of parameters are selected randomly in each iteration to form the tree. This is done to avoid overfitting. The default boosting algorithm ‘gdbt’ is used along with a binary classification application. The rest of the parameters are left to their default settings. For training the model we use both word embeddings and character embeddings to create the train, test and validation features. This also handles the occurrence of misspelled/custom words in our dataset.

1.5 Performance Evaluation

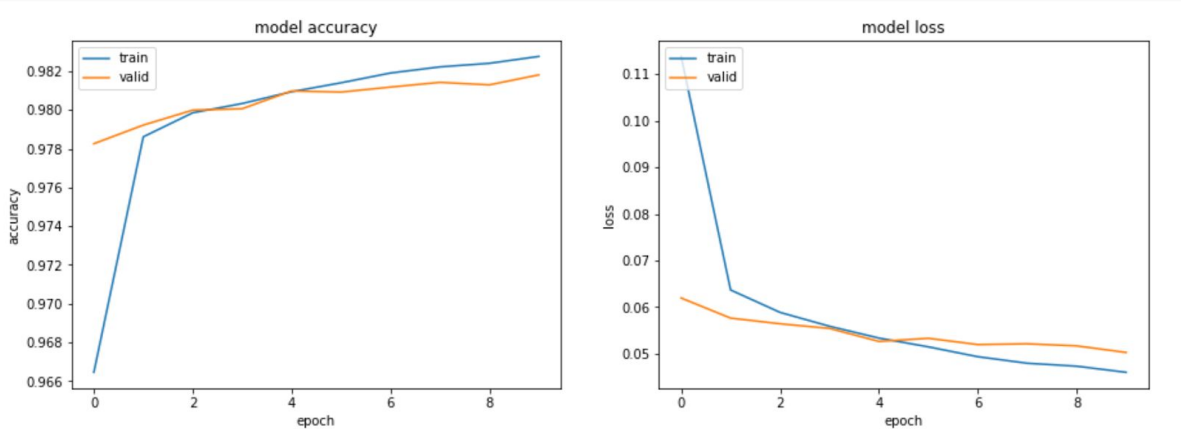
1.5.1 Model Evaluation/Validation

Model	ROC AUC Training Set	ROC AUC Validation Set	ROC AUC Testing Set
Multinomial Naive Bayes	X	0.5480	0.5458
Logistic Regression	X	0.7340	0.7437
Naive Bayes Logistic Regression	X	0.5476	0.5383
RNN (LSTM)	0.9725	0.9803	0.9801
CNN	0.9846	0.9746	0.9434

GloVe + RNN	0.9889	0.9851	0.9756
LGBM	0.9924	0.8030	0.6223

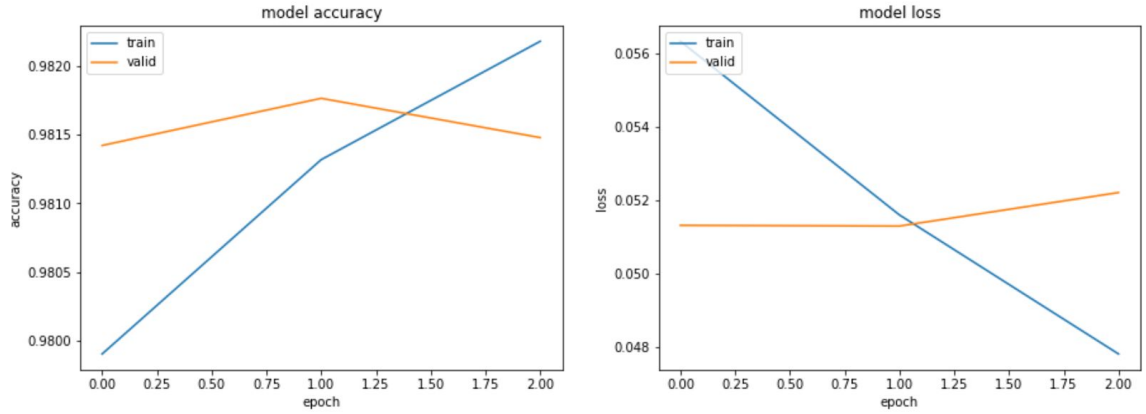
The Logistic Regression model (Naive Bayes) is chosen as benchmark model, The model showed good results by applying TF-IDF Vectorizer with n-grams. As discussed in section 1,4.1, the TF-IDF Vectorizer are word frequency scores that try to highlight words that are more interesting. We observe that the vanilla Logistic Regression model in fact has a higher ROC AUC score than the Multinomial Naive Bayes and Logistic Regression (Naive Bayes) models. In the comments classification problem, certain keywords in a sentence can simply place it into a subtype of toxicity. TF-IDF helps to highlight these keywords. It is shown that RNN (LSTM and vanilla) is much better than CNN in toxicity classification. CNN is good at extracting position invariant features and RNN good at modeling units in sequence. In this project, the toxicity can be either determined by some key words or more often by the long-range semantics, thus RNN performs better. By adding the GloVe embedding layer, the performance was further improved. Because the word embedding layer was trained with a more reasonable initializer. LGBM on the other hand lies somewhere in the middle in performance between our baseline models and neural networks. The RNN with LSTM architecture performs best with an ROC AUC score of 0.9801 on the test dataset.

From the below visualization, we observe that the CNN model is quickly overfitting in 2 epochs. The RNN model with GloVe embeddings performs the best and slightly overfitted around 6 epochs. Whereas the RNN model with LSTM is under-fitting in 2 epochs and should be run for more number of epochs to get the ideal fitting. As compared to our baseline models of Naive Bayes - Logistic Regression, neural networks performs better as expected.

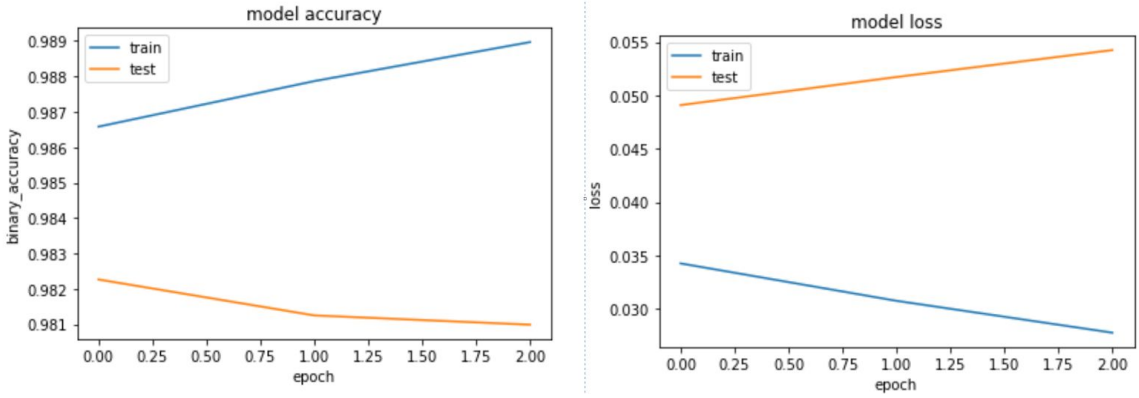


RNN GloVe model accuracy and model loss

CNN model accuracy and model



RNN with LSTM



1.6 Conclusion

In this work we presented multiple approaches for toxic comment classification. We can conclude that some models such as shallow learners with deep neural networks are especially effective in this task. Forming an ensemble learner of these neural networks would potentially provide the ideal output. Minor architecture changes made very little impact on AUC ROC score, since most of the model complexity lay in the pre-trained embeddings. The overall score of the model barely changed post adding dense layers, gaussian vs spatial dropout, etc. For the performance comparison with respect to CNN and RNN, the order of the words mattered. The same phrase with two words swapped can mean completely different things. This indicates that as CNN approaches rely on max-pooling as a crutch, they are difficult to work with.

Due to limited time and computation resources, there are some other approach that can be experimented later for further improvement such as diverse pre-trained word embeddings, translation as train/test - time augmentation. What also needs to be delved deeper into is the possible presence of doubtful labels, the presence of a toxic comment without there being a swear word in a sentence such as “she looks like a horse”, metaphors etc.

Appendix [Folder Structure]

The submission zip contains the following items. The scripts are provided in the scripts folder as for data_cleaning, data exploration, data preprocessing and model training. The pdf copy of the .ipynb notebook files is present in the Notebook folder.

Scripts Folder

data_cleaning.py : code to clean the text comments like removing punctuation and typos/spelling mistakes.
data_preprocessing.py: code to apply a tokenizer to the cleaned text to encode the text into a sequence of numbers
data_exploration.py: code for some exploratory research to better understand the data set.
glove_rnn.py: code for model training and evaluation of GloVe+RNN
logistic_regression.py: code for model training and evaluation of the benchmark model
rnn_lstm.py: code for model training and evaluation of RNN with LSTM
cnn.py: code for model training and evaluation of CNN
lgbm.py: code for model training and evaluation of LGBM

Notebooks Folder: This folder contains the the .ipynb files. These include cnn_rnn_GloVe.pdf, LGBM.ipynb, LR.ipynb, rnn_lstm.pynb

Figures Folder: This folder contains the model evaluation figures
Report.pdf: Final Project Report

References

1. van Aken, Betty, et al. "Challenges for Toxic Comment Classification: An In-Depth Error Analysis." *arXiv preprint arXiv:1809.07572* (2018).
2. https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568
3. https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline
4. https://medium.com/@nupurbaghel/toxic-comment-classification-f6e075c3487a
5. https://medium.com/@nehabhangale/toxic-comment-classification-models-comparison-and-selection-6c02add9d39f
6. https://medium.com/@sabber/classifying-yelp-review-comments-using-cnn-lstm-and-pre-trained-glove-word-embeddings-part-3-53fcea9a17fa
7. https://www.kaggle.com/peterhurford/lightgbm-with-select-k-best-on-tfidf
8. https://nycdatascience.com/blog/student-works/toxic-comment-classification-challenge-a-kaggle-competition/
9. https://www.aclweb.org/anthology/W18-5105
10. https://medium.com/@zake7749/top-1-solution-to-toxic-comment-classification-challenge-ea28dbe75054