1. Definition

The internet can feel like a safe platform for communication. It strips the need for a face-to-face conversation. It enables us to be identified from a random nickname or profile picture and keeps our true identities hidden from others. However, it is these characteristics of the internet that has made it a far from safe place to hold public conversations.

The increasing degree of conversational toxicity and online bullying by 'keyboard warriors' is concerning. The threat of online abuse and harassment can hinder people from genuinely expressing themselves and seeking opinions from others. In more severe cases, it can negatively impact the mental and psychological well-being of those being bullied online. It is therefore of value to be able to effectively identify toxicity in online postings, especially user comments. It enables platforms to facilitate healthier and safer online discussions for their users, which is a win-win situation for both users and the platforms.

In this project, we will be building multi-headed machine learning classification models to detect toxicity in online comments. There are 6 class labels for classification of comments, namely, toxic, severe toxic, obscene, threat, insult and identity-hate. Specifically, the strategy would be to build different models and make predictions separately for each class of toxic comments. This would be equivalent to performing a binary classification for each class.

This project leverages the dataset used in Kaggle's Toxic Comment Classification Challenge[1], provided by the Conversation AI team, a research initiative founded by Jigsaw and Google. The dataset consists of comments from Wikipedia's talk page edits. There are 159,571 training examples and 153,164 testing examples. Out of the testing examples, only ~42% of them have labels and would be used for scoring and evaluation. The data dictionary can be found in Table 1. Mean column-wise ROC AUC would be used as the evaluation metric, adapted from the evaluation criteria of the Kaggle challenge itself. This would be computed by the average of AUC scores of the 6 class labels.

Data	**Description**	**Schema**
train.csv (159,571 rows x 8 fields)	Wiki talk page edit comments in the train dataset Note that examples where all class labels take value 0 are considered non-toxic	- id: str - comment_text: str <u>class labels</u> - toxic: int (0 or 1) - severe_toxic: int (0 or 1) - obscene: int (0 or 1) - threat: int (0 or 1) - insult: int (0 or 1) - identity_hate: int (0 or 1)
test.csv (153,164 rows x 2 fields)	Wiki talk page edit comments in the test dataset	- id: str - comment_text: str
test_labels.csv (153,164 rows x 7 fields)	Labels for test dataset Note that if all class labels are -1, it is not used for scoring and evaluation	- id: str - toxic: int (0 or 1 or -1) - severe_toxic: int (0 or 1 or -1) - obscene: int (0 or 1 or -1) - threat: int (0 or 1 or -1) - insult: int (0 or 1 or -1) - identity_hate: int (0 or 1 or -1)

Table 1. Data dictionary of dataset

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

2. Analysis

2.1 Exploratory Data Analysis (EDA)

1. Missing and duplicated values check

   There are no null values in both train and test datasets. It is also observed that there are no duplicated ids in both datasets.

```
Train data: (159571, 8)

No. of duplicated ids: 0
No. of missing values
id               0
comment_text     0
toxic            0
severe_toxic     0
obscene          0
threat           0
insult           0
identity_hate    0
dtype: int64
```

```
Test data: (153164, 2)

No. of duplicated ids: 0
id              0
comment_text    0
dtype: int64
```

Figure 1. Train and test data overview

2. Class distribution

   With reference to Figure 2, there is severe class imbalance in the train dataset, with non-toxic comments (comments with a value of 0 across all the 6 toxic class labels) forming the majority, almost 90% of all comments. Threat comments form the smallest class with 478 training examples.

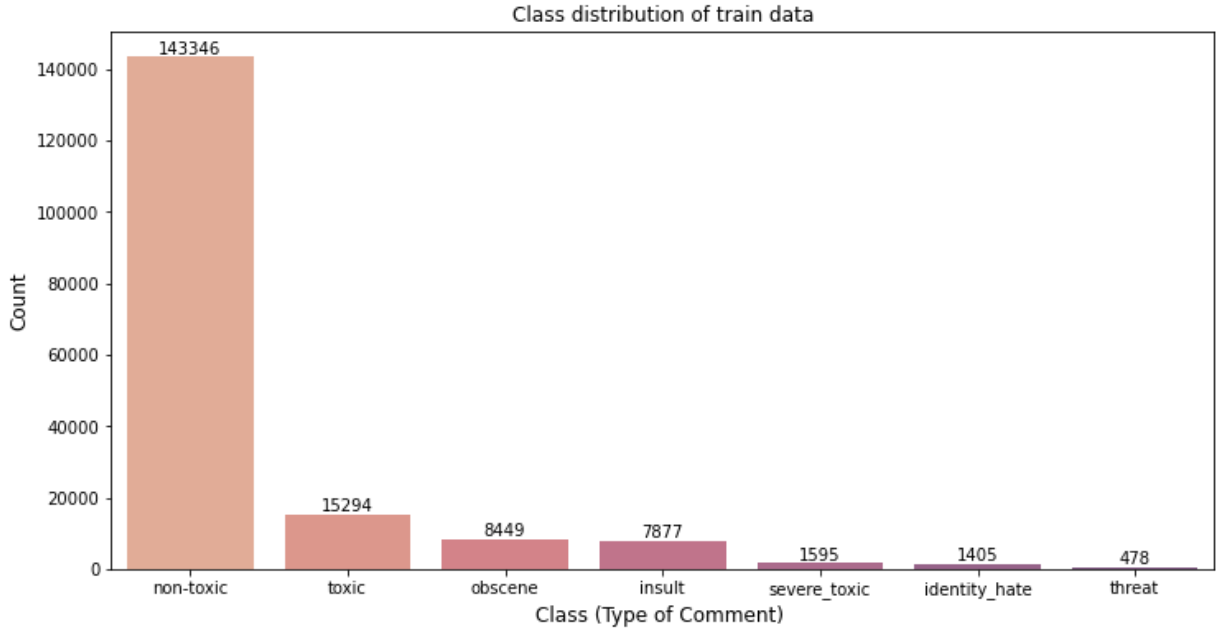

Figure 2. Class distribution of train data

3. Multi-class labels

With reference to Figure 2., there appears to be more class labels (178,444) that number of training examples (159,571). This is because one text comment can be attributed to more than toxic class label. With reference to Figure 3., excluding non-toxic comments i.e. those with 0 labels, most of the training examples have 3 or less class labels. It is interesting to note that there are 31 training examples that have a label of 1 for each of the 6 toxic class labels.

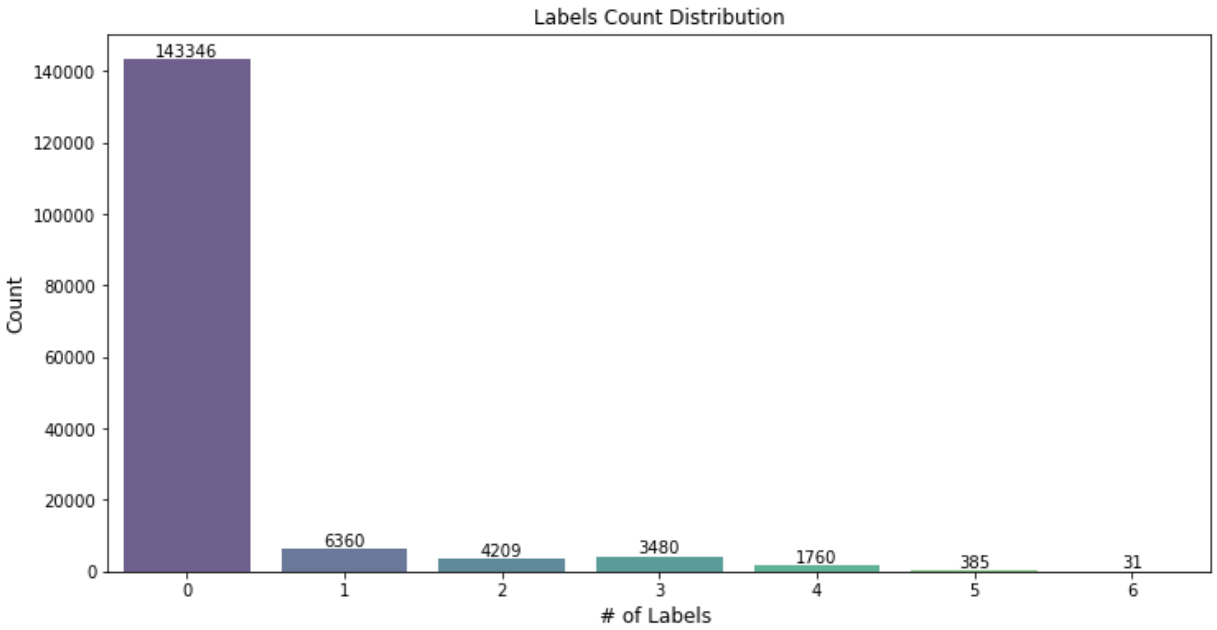

Figure 3. Multi-class label distribution

4. Correlation

Cramer's V statistic is often used as a measure of association between a pair of categorical variables with a value between 0 and 1 (inclusive). The larger the Cramer's V, the higher the correlation between the variable pair. In this project, Cramer's V was used to compute correlation between the class labels. It was observed that the correlation between 'threat' and other class labels are generally low at less than 0.2. This is in line with intuition as well because not every toxic comment would contain an explicit statement of intention to cause harm or damage to others. The highest correlation of 0.741 was observed between 'obscene' and 'insult' class labels.

5. Word clouds

The word clouds in Figure 4. show top 2000 words that appear in each of the various classes. The non-toxic word cloud contains many neutral sounding words and verbs. In contrast, the word clouds of the 6 toxic classes contain vulgarities and many words with negative connotation. There is also some degree of overlap of most commonly occurring words in the 6 toxic word clouds. This could imply that the same words used in different context can constitute a different type(s) of toxicity.

Figure 4. Wordclouds

6. Content of comments

In this last part of EDA, simple features were created to answer the following 3 questions:

1. Does length of comment affect toxicity?

   Length of comment was looked at from 3 measures, character, word and sentence count. With reference to Figure 5, there does not appear to be as strong correlation between length of comment and whether a comment would be toxic in nature. This can be observed from the

blue and yellow violin plots, whose shape do not differ significantly from each other for each measure of length.

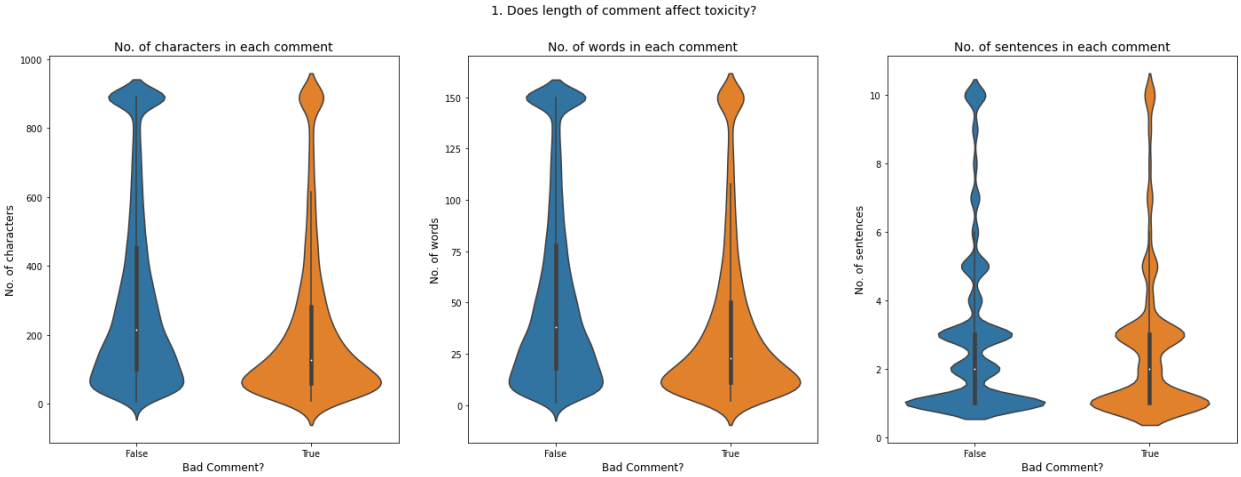

Figure 5. Violin plots of length vs nature of comment

2. Do unique words affect toxicity?
   With reference to the left subplot in Figure 6., it can be observed that the distribution of unique words across 'Non-toxic' and 'Bad' comments are quite similar. An exception is observed at low % of unique words, where a slight bump is present for 'Bad' comments. This implies that there is a larger proportion of comments where text is repeated many times and not meaningful a.k.a spam.

3. Do uppercase words affect toxicity?
   With reference to the right subplot in Figure 6., the distribution of uppercase words across 'Non-toxic' and 'Bad' comments are largely similar, with majority of the comments having 20% or less uppercase words. A bump can also be observed at high % of uppercase words, with that for ‘Bad’ comments being more pronounced than ‘Non-toxic’ comments.

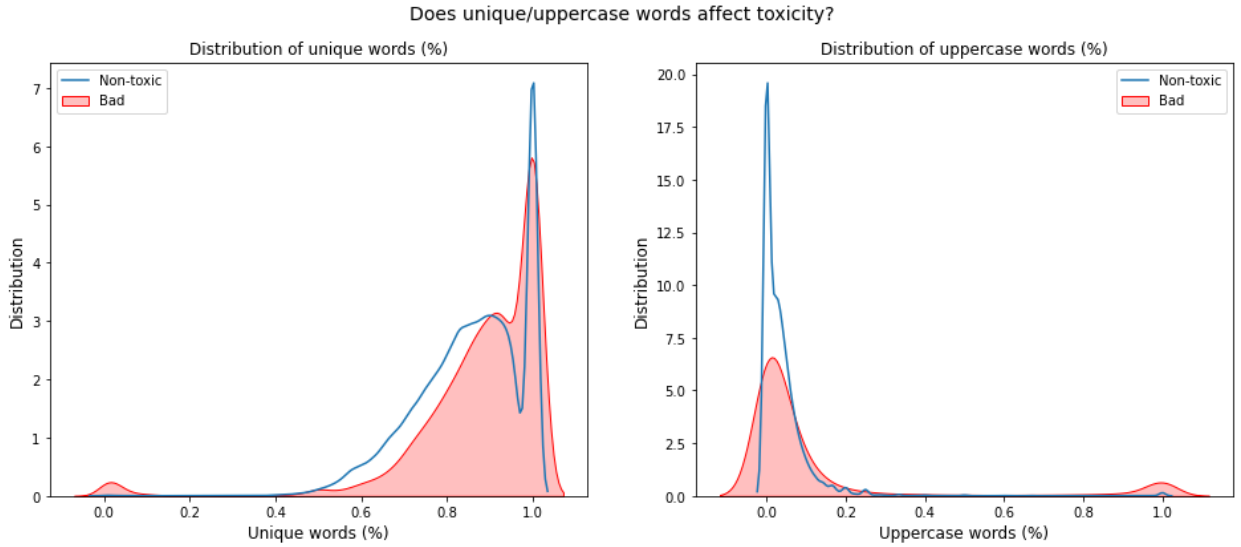

Figure 6. Distribution plots of unique words (%) and uppercase words (%)

2.2 Algorithms and Techniques

Logistic Regression

Logistic Regression is a linear model that can be used to predict binary classes. It is a special case of Linear Regression where the output variable takes binary values. It predicts the likelihood of occurrence of the default class (i.e. 1), transformed by the logistic/sigmoid function, to a value between 0 and 1. The coefficients of the Logistic Regression equation is learned and tuned from a process known as maximum likelihood estimation (MLE), which minimizes the error between the actual and predicted values over training iterations.

As the strategy in this project is to perform binary classification of each of the 6 classes separately, Logistic Regression would be a suitable algorithm to start with. In addition, it is also straightforward to set up and capable of producing reasonable results. Thus, it was selected as the baseline model.

NB-SVM

The next algorithm used in this project is the NB-SVM algorithm proposed by Wang & Manning in 2012. Multinomial naïve Bayes (MNB) performs very well on short snippets of text like comments or tweets, while support vector machines (SVM) fair better on longer documents. The authors combined these 2 models into what is known as the NB-SVM algorithm, by coupling SVM with naïve Bayes probabilities as features.

This algorithm also does not require complex or deep feature engineering and works well with bag-of-words type features. Furthermore, it was originally developed for sentiment and topic classification of texts. Thus, the context in which this algorithm was originally developed for matches the context of this text classification problem very well.

Neural Network – RNNs

Recurrent neural networks (RNNs) are a class of neural networks with internal memory. When making a prediction, it would consider not only the current input, but also what it has learned from all previous output. It is most suitable for time series analysis or processing of sequence data. Text is an example of sequence data and thus RNN would be a suitable neural network architecture for this problem.

One main problem with RNNs is its short-term memory caused by the vanishing gradient problem. Gradients are shrunk due to the multiplication of gradients across layers (which can be exponentially decreasing) during backpropagation. As the value of gradient approaches 0, it will not contribute much to the learning process, and the network weights will not be updated. This is more common for earlier layers, and what this means is that RNN can lose important information from earlier parts of a long sequence, thus having short-term memory.

LSTM and GRUs were developed to address RNN's short term memory problem by using gates that regulate the flow of information. These gates pass on relevant information down sequences by learning which part of a sequence should be kept or thrown away. One key difference between LSTM and GRU is the number of gates. LSTMs have 3 gates, the forget, input and output gates while GRUs have 2 gates, the update and reset gates. In addition, LSTMs have cell states, which serves as the memory network to transfer information down the sequence chain. In contrast, GRUs rely on hidden states to transfer information.

3. Methodology

The common preprocessing steps applied across models are to convert text to lowercase, tokenize the text and strip or split text based on defined punctuation characters. Additional preprocessing was also performed for some models by removing English stopwords from the NTLK corpus, performing lemmatization using WordNet lemmatizer as well as some rules to remove text that do not contribute to the toxicity of a comment such as website URLs, IP address and usernames.

The metric used for model evaluation is the mean column-wise ROC AUC, which was adapted from the evaluation criteria of the Kaggle challenge itself. This would be computed by the average of AUC scores of the 6 class labels.

The main libraries used for training and prediction are sklearn as well as tensorflow-keras. For each type of algorithm, a run() function was defined that chained the pipeline of steps to be performed for 1 cycle of modelling and prediction. A seed was set wherever possible to ensure reproducibility of results. Unfortunately for RNN modelling, as tensorflow-keras was ran on GPU, it was challenging to set a seed for reproducibility.

Logistic Regression baseline model was ran with bag-of-words type features. As for NB-SVM algorithm, TF-IDF type features was explored to see if it would contribute to an improved model performance. Here, unigram and bigram bag-of-word type as well as TF-IDF features were created based on the training data, which was then used to fit and transform both training and test data sets. Additional models for NB-SVM based algorithm were also built to evaluate the impact of stopwords removal and lemmatization on model performance.

With regards to neural network modelling, LSTMs were used as a baseline architecture for neural networks. GRUs based neural networks were also built and compared against LSTMs' performance. Both were built with bidirectional 2 layer network configuration with a dropout of 0.1. It was also of interest to explore if the model would benefit from being initialized with a pre-trained word vector. 2 types of pre-trained Glove word vectors were used, 1 trained based on Wikipedia corpus, and the other based on Twitter tweets. It is hypothesized that tweets are more similar to the text comment dataset in terms of how a piece of information is being communicated e.g. shorter text length, imperfect grammar, urban dictionary lingo etc. Thus, the model initialized with Twitter pre-trained vector is likely going to give a better model performance than that initialized with Wiki corpus pre-trained vector. A final model was built that takes in rules to remove texts that are not considered to be strong in contributing to toxicity of comments such as URLs, IP addresses and usernames.

4. Results

Table 2. shows the mean ROC-AUC scores of the various models built in this project. The baseline Logistic Regression model with bag-of-word type features gave the lowest score. Switching the algorithm to NB-SVM saw model performance improve by 0.4%. This implies that the baseline model is a rather strong performing one.

The more significant model improvement came from the use of TF-IDF type features, which led to an increase of mean ROC-AUC score by 3.6% for the NB-SVM model (S/N 2 to S/N 3). This is likely due to the fact the TF-IDF helped with downplaying the importance of commonly occurring words that appear across the different text comments, giving more weight to more unique and less frequent words. It is also observed that the introduction of additional preprocessing steps did not seem to alter model performance significantly. For the NB-SVM model, the best performance came from TF-IDF representation, coupled with lemmatization – S/N 5 highlighted in yellow.

It is interesting to note that the initial LSTM and GRU models built did not perform better than the best NB-SVM model. Although the use of pre-trained word vectors helped to better the performance of GRU model, the delta improvement was not significant. In addition, the results appear to be in line with the hypothesis that pre-trained twitter vector gave better performance than pre-trained Wikipedia corpus vector, albeit by a small percentage of 0.1%. The best neural network model came from GRU initialized with pre-trained twitter vector with additional preprocessing rules.

Some possible reasons for the rather comparable performance of NB-SVM and LSTM/GRU models include:

- LSTM and GRU models were trained on only 2 epochs, while the NB-SVM models were programmed to run for 200 iterations. With more training iterations for the neural network models, their performance might improve.
- Neural networks work better for longer text like documents, articles and blogs, while NB-SVM really shines in the context of shorter text like comments.

S/N	Model	Description	Mean ROC-AUC
1	LR-CntVec (Baseline)	- Bag of features representation - Logistic Regression model	0.93805
2	NB-SVM-CntVec	- Bag of features representation - NB-SVM Model	0.94214
3	NB-SVM-TfIdf	- TF-IDF representation - NB-SVM Model	0.97630
4	NB-SVM-TfIdf-StopW	- S/N 3 with stopwords removal	0.97373
5	NB-SVM-TfIdf-Lemma	- S/N 3 with lemmatization	0.97728
6	NN-LSTM	- Bidirectional LSTM model of 2 fully connected layers with dropout	0.97413
7	NN-GRU	- Bidirectional GRU model of 2 fully connected layers with dropout	0.97458
8	NN-GRU-Glove-Wiki	- S/N 7 initialized with glove vector pre-trained on Wikipedia corpus	0.97837
9	NN-GRU-Glove-Twitter	- S/N 7 initialized with glove vector pre-trained on Twitter tweets	0.97960

10	NN-GRU-Glove-Twiter-PreP	- S/N 9 with additional preprocessing performed to remove text such as URLs, IP addresses and usernames	0.97984

Table 2. Results comparison table

5. Conclusion

In conclusion, this project was a great opportunity to be involved in the entire process of a data science/machine learning project, from sourcing of data to data exploration and analysis, followed by modelling and evaluation. Specifically, three types of algorithms were used across 10 different models with differing pre-processing and/or initialization methods in this project. The best neural network model came from GRU initialized with word vector pre-trained on Twitter tweets with additional preprocessing rules to remove text like website URL, IP addresses and usernames. Further enhancements to this project include using state-of-the-art pre-trained embeddings like BERT in the modelling process. In addition, hyperparameter tuning can be performed to find the best set of model parameters that optimize model performance.