Toxic Comment Classification

COMP9417 Project

Mimi Tran
z5158339

Karim Saad
z5209523

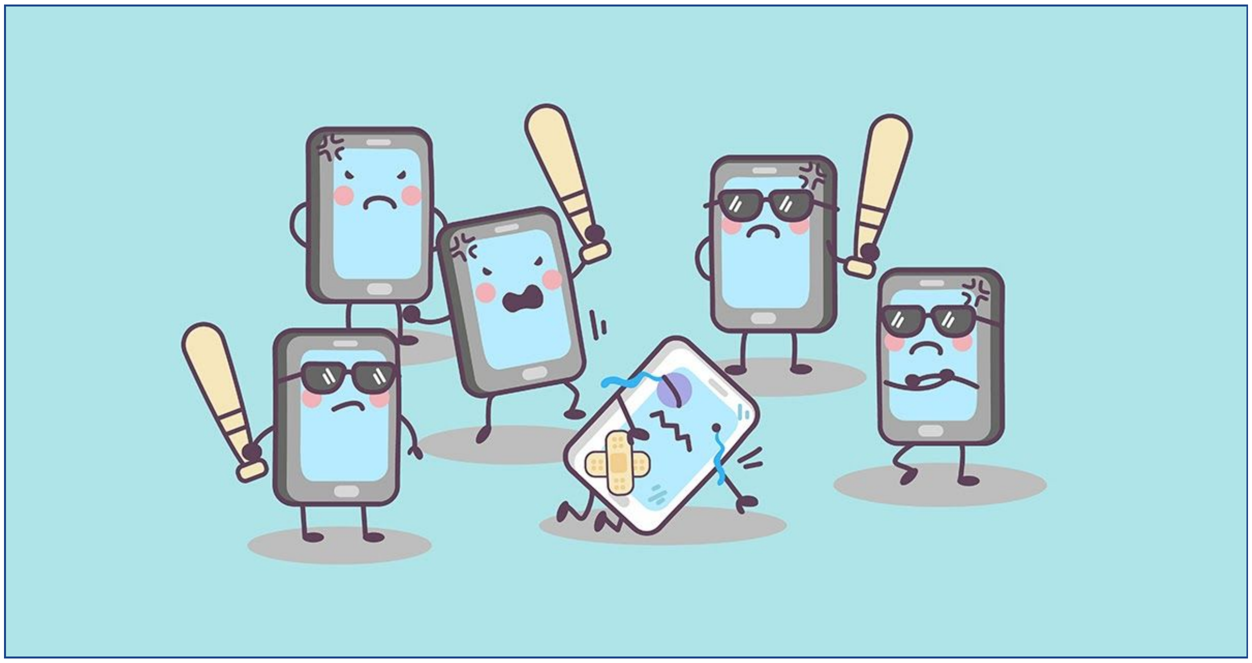

Table of Contents

Project Description

The project we have chosen to undertake is Toxic Comment Classification. This is a challenge posted by Jigsaw/Conversation AI on **Kaggle**. This project appealed to us as our generation frequently uses social media, where various platforms allow multiple users to interact and communicate freely. These platforms often do a poor job of filtering and restricting toxicity, which can heavily impact the user friendliness of the platform.

The aim is to build a model that allows us to determine the toxicity of a given comment. A unique requirement of this project is the ability to distinguish between various categories of toxicity. This will allow platforms to permit certain types of toxicity (i.e. obscenity), but not other types (i.e. identity hate).

Exploratory Data Analysis

Prior to implementing our model, we need to conduct some preliminary data analysis. This provides insight and a visual representation of the data provided, and is imperative to ensuring the models are built correctly.

The data contains the following information for each comment;

- id - unique string for each comment
- comment_text - body of text that the user writes
- toxic, severe_toxic, obscene, threat, insult, identity_hate - 0 or 1 to determine the toxicity classification for that comment

The train dataset contains 159,571 comments, and the test dataset contains 153,164 comments, resulting in a train to test ratio of 51:49, a fairly even distribution.

Note: The classifications for each of these comments was crowdsourced, so they may be mislabelled, causing some unwanted noise.

Despite the train dataset containing 159,751 comments, this doesn't mean that all of them have one of the six toxicity classifications. In fact, there are 143,346 clean (non-toxic) comments. The distribution of each toxicity classification is given below (*Figure 1*).

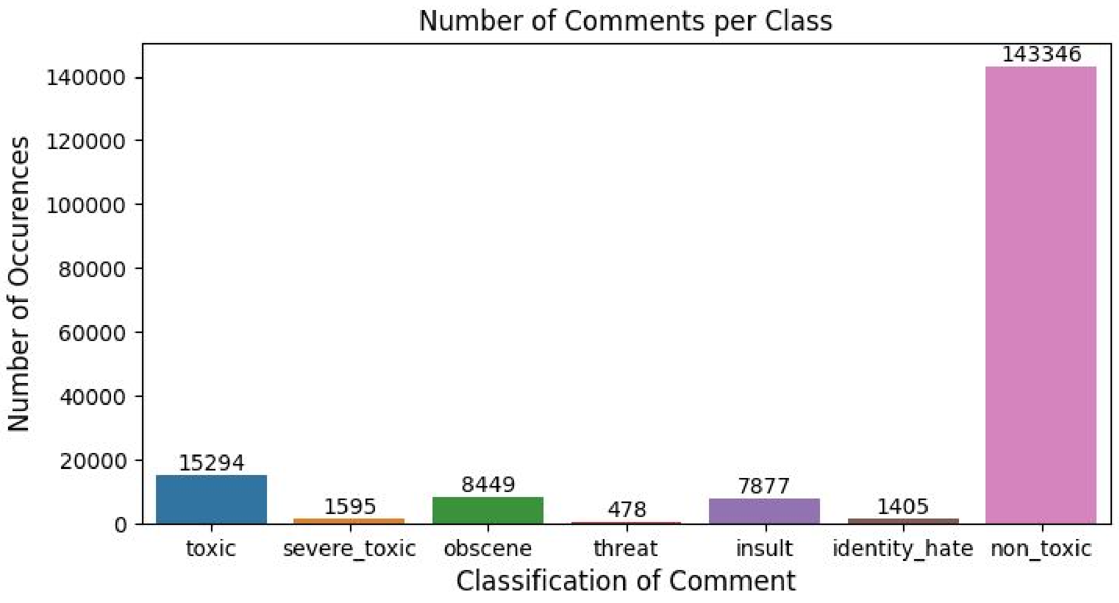

Figure 1

The above distribution indicates that there are 35,098 toxicity tags. However, since there are 143,346 clean comments, and 159,571 total comments, this indicates that there are several comments tagged with more than one toxicity classification. The distribution indicating the number of comments containing 1 or more toxicity tags is given below (*Figure* 2).

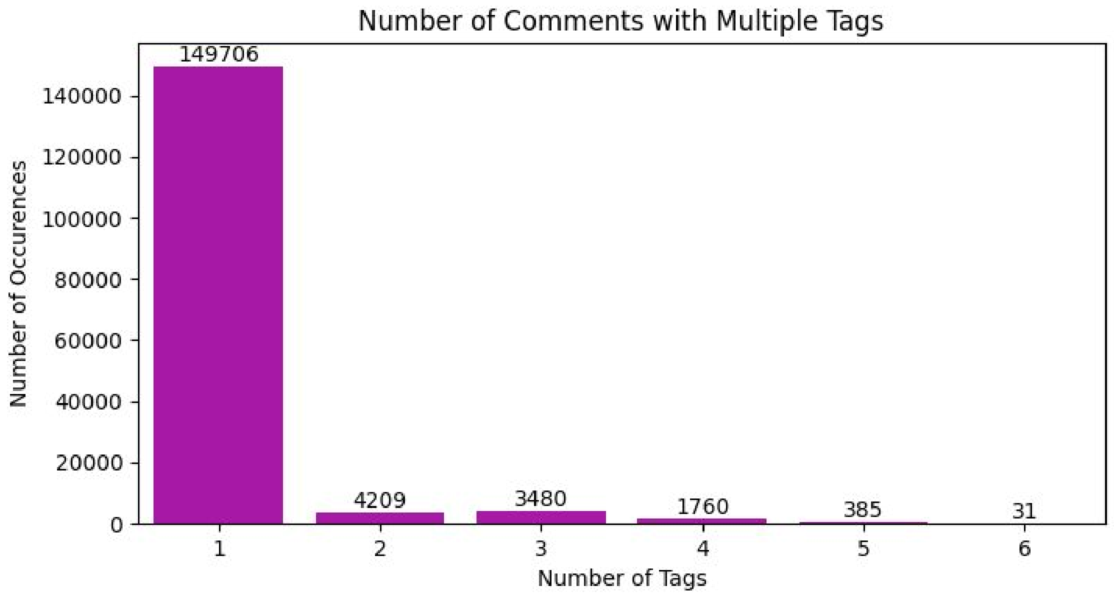

Figure 2

The above distribution indicates that most toxic comments only have one toxicity classification, but 9,865 comments have multiple toxicity classifications. The heatmap below shows the correlation between two toxicity classifications (*Figure* 3).

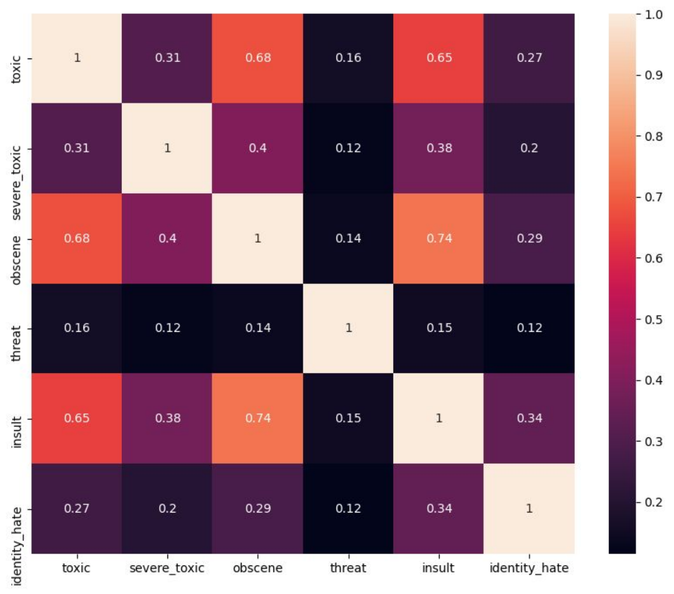

Figure 3

For each toxicity classification, there is also a word cloud attached, allowing a visual representation of the most common words associated with each type of toxicity.

WARNING: Due to the nature of this task, the language used in these comments, and displayed in the word clouds may be offensive. Please do not open these images if you aren't comfortable with vulgar, discriminatory or obscene language.

Implementation

Long Short-Term Memory (LSTM)

The purpose of using the Keras library is to reduce the amount of structure handling, indexing and slicing required while developing our neural learning model. Due to the high level implementation of many neural learning concepts, it's imperative that each step is taken deliberately, and all parameters and functions are chosen with intent.

In order to feed our data to the LSTM, we need to provide it purely with words. Fortunately, Keras has an in-built function to tokenize[1] the comments.

Following tokenization, the comments are now of inconsistent length, as each comment has a unique amount of words. However, this is not suitable to feed into the LSTM model, as it requires a stream of data with consistent length. The data is then padded[2], selecting an appropriate maximum length so as to not trim excess words.

Below (*Figure* 3) is a general architecture for the implementation, including shapes. This will provide a visualisation of the flow of the model, as well as assist in debugging later down the line.

Note: Some parameters will be changed during tuning stages, but this diagram will hold true for the standard parameters.

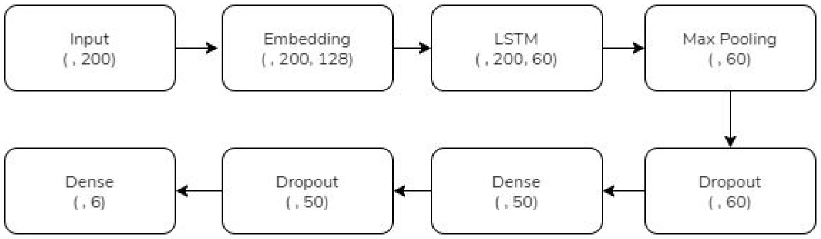

Figure 3

The input[3] layer passed into the model is a list of sentences, each with a dimension of 200 (number of words/features).

The embedding[4] layer now projects the words to a new vector space (defined by the embedding size). However, in order to do so, the words must be fed as numerical vectors. Fortunately, this data is already in numerical vectors following the tokenization step. The output of the embedding layer becomes a list of coordinates which each feature (word) holds in the new vector space.

Now, the resulting tensor is fed to the LSTM[5] layer. LSTM, similar to Recurrent Neural Networks (RNN), recursively feeds the output of the previous layer into the current layer. An accurate visual representation for LSTM is given below (*Figure* 4).

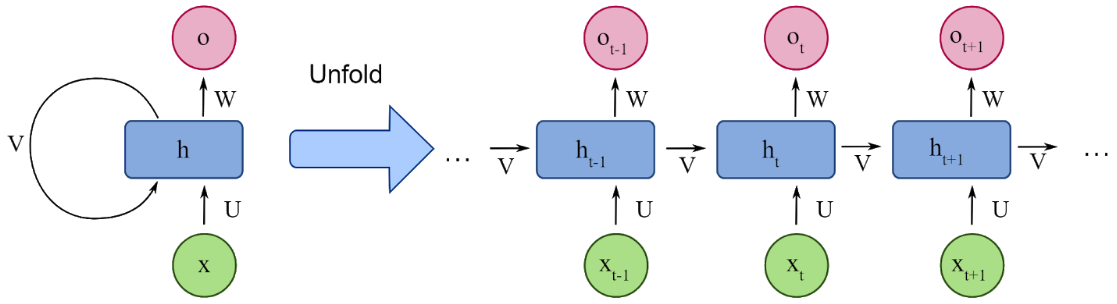

Figure 4

In order to reduce the dimensionality of the LSTM output to a 2D tensor from a 3D tensor, a pooling layer is implemented. The most traditional pooling method used for Convolutional Neural Networks (CNN) is max pooling[6], which will take the maximum value of each patch. This method of dimensionality reduction avoids excessive disruption of the original data.

The original datasets provided are complete, and not have any missing data (as shown in the Exploratory Data Analysis). In order to provide the model with an apt opportunity to handle missing data, and generalise appropriately, a dropout[7] layer is used. Ten percent (10%) of the data is chosen indiscriminately and 'disabled' for the next layer.

Prior to the data being passed onto the next layer, an activation[8] layer is used to determine which neuron inputs will be passed on. In this implementation, a dense RELU layer is used.

The second dropout layer is again used for a similar purpose, to ensure that the model is able to train with minimal and incomplete data.

The final step of the model is to classify the data to each of the six labels. A dense Sigmoid layer is used here, due to the binary classification required for each label.

The model is then compiled, using binary cross entropy to calculate the loss of the model (due to the binary classifications required), and the Adam[9] optimizer (stochastic gradient descent method).

Finally, the model must be fit to the training data. The parameters for this function are the most impactful to the accuracy and loss of the model, so these parameters are fine tuned and selected for optimal performance. The classifications for the test dataset are available in the appendix as .csv files. The loss and accuracy scores are documented in the results section.

Naïve Bayes - Support Vector Machine (NB-SVM)

Naive Bayes (NB) and Support Vector Machine (SVM) models can be used as effective baseline methods for text classification. In this implementation, a logistic regression[10] model has been chosen, as it runs faster than a standard SVM[11]. In practice, when comparing cross validation scores, they produce very similar results.

In this model, Document Term Matrix is used as an implementation of the **Bag of Words** concept, where each row of the matrix is a document vector, with one column for every term in the entire corpus. We have used TfidfVectorizer[12] that converts a collection of raw documents into a collection of TF-IDF features, which is is a convenient input for logistic regression:

Document No	Word No	TF-IDF
0	22653	0.11225677989510632
0	35387	0.11974559252684826
: :	: :	: :
159570	6837	0.03184713429946594
159570	247352	0.05047743493072502

In this model, the **Naive-Bayes** feature equation has been used, where the **count vectors** are defined as:

$$p = \alpha + \sum_{i: y^{(i)} = 1} f^{(i)}$$

$$q = \alpha + \sum_{i: y^{(i)} = 0} f^{(i)}$$

where:

- $\mathbf{y^{(i)}}$ represents a label ∈ {0, 1}
- $\mathbf{f^{(i)}}_j$ represents the number of occurrences of feature $\mathbf{V}_j$ in training case **i**
- α is the smoothing parameter (in our case, we have picked $\alpha = 1$)

And hence, the **log-count ratio** is:

$$\mathbf{r} = \log\left(\frac{\mathbf{p}/||\mathbf{p}||_1}{\mathbf{q}/||\mathbf{q}||_1}\right)$$

[13]

To execute the **NB-SVM** model, x is assigned to the element-wise product of $\widehat{r}$ and $\widehat{f}^{(k)}$:

$$x = \widehat{r} \circ \widehat{f}^{(k)}$$

Results

The cross validation scores[14] obtained from the NB-SVM model:

Class Name	Cross Validation Score
Toxic	0.9594
Severe Toxic	0.9906
Obscene	0.9776
Threat	0.9974

Insult	0.9721
Identity Threat	0.9922
Total (Average)	0.9815

The accuracy and loss scores obtained from the LSTM model:

Note: Standard Parameters used - *batch_size=32*, *epochs=2*, *validation_split=0.1*

Parameter Changed	Loss	Accuracy
Standard Parameters	0.0505	0.9938
Higher Batch Size	0.0514	0.9942
Lower Batch Size	0.0508	0.9940
Higher Epochs	0.0456	0.9930
Lower Epochs	0.0525	0.9941
Higher Validation Split	0.0517	0.9942
Lower Validation Split	0.0517	0.9942
Bidirectional	0.0504	0.9933

Discussion

Comparison of Results

Using the NB-SVM implementation, a training accuracy of about 98.1% was achieved. However, using the Neural Network implementation, a training accuracy of about 99.4% was achieved. This difference, while insignificant, can still be explained by the structure of the data, as well as the advantages and disadvantages of each different model.

Generally, neural networks will perform better than SVM as long as certain conditions are held. If the data is too noisy, neural networks tend to overfit the data, and begin to recognise the noise as a feature of the pattern. However, the noise in this dataset is only attributed to mislabelling by crowdsourcing, resulting in a fairly low amount of noise.

Another consideration is load and time. Neural networks output high accuracy results, however they take significantly longer to train, fit, evaluate and predict.

Future Work

A plethora of varying cross validation methods[15] should be experimented with in the future. In this NB-SVM model, 5-fold cross validation (the default option) has been selected. In the future work, Stratified K-fold cross validation method be selected, as it randomly shuffles the data prior to dividing it into folds. Leave-One-Out cross validation is an exhaustive method, as it validates the model on every possible combination of data points. This may not be ideal for this implementation, as it can be very costly to run with the large amount of data that we have.

Ensemble learning is a facet of machine learning that was not explored in this project. In the future, several models, including the two that were implemented here, may be combined to provide a more accurate classification on the test dataset. The weights attributed to the results of each implementation will have to be carefully researched and considered.

The dataset provided may also hold some issues. Due to the nature of this data, there may be an excessive amount of ‘spamming’ where certain users will post several comments of the same format. This may skew the data if not appropriately handled, and reduce classification accuracy. Several feature engineering techniques can be applied to clean the corpus.

References

[1] TensorFlow. 2020. Tf.Keras.Preprocessing.Text.Tokenizer | Tensorflow Core V2.3.0. [online] Available at: <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer> [Accessed August 2020].

[2] TensorFlow. 2020. Tf.Keras.Preprocessing.Sequence.Pad_Sequences | Tensorflow Core V2.3.0. [online] Available at: <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences> [Accessed August 2020].

[3] TensorFlow. 2020. Tf.Keras.Input | Tensorflow Core V2.3.0. [online] Available at: <https://www.tensorflow.org/api_docs/python/tf/keras/Input> [Accessed August 2020].

[4] Keras.io. 2020. Keras Documentation: Embedding Layer. [online] Available at: <https://keras.io/api/layers/core_layers/embedding/> [Accessed August 2020].

[5] Keras.io. 2020. Keras Documentation: LSTM Layer. [online] Available at: <https://keras.io/api/layers/recurrent_layers/lstm/> [Accessed August 2020].

[6] TensorFlow. 2020. Tf.Keras.Layers.Globalmaxpool1d | Tensorflow Core V2.3.0. [online] Available at: <https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool1D> [Accessed August 2020].

[7] Keras.io. 2020. Keras Documentation: Dropout Layer. [online] Available at: <https://keras.io/api/layers/regularization_layers/dropout/> [Accessed August 2020].

[8] Keras.io. 2020. Keras Documentation: Layer Activation Functions. [online] Available at: <https://keras.io/api/layers/activations/> [Accessed August 2020].

[9] Keras.io. 2020. Keras Documentation: Adam. [online] Available at: <https://keras.io/api/optimizers/adam/> [Accessed August 2020].

[10] Scikit-learn.org. 2020. Sklearn.Linear_Model.Logisticregression — Scikit-Learn 0.23.2 Documentation. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html> [Accessed August 2020].

[11] Scikit-learn.org. 2020. Sklearn.Svm.SVC — Scikit-Learn 0.23.2 Documentation. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html> [Accessed August 2020].

[12] Scikit-learn.org. 2020. Sklearn.Feature_Extraction.Text.Tfidfvectorizer — Scikit-Learn 0.23.2 Documentation. [online] Available at:

<https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html> [Accessed August 2020].

[13] Wang, S. and Manning, C., 2020. [online] Nlp.stanford.edu. Available at: <https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf> [Accessed August 2020].

[14] Scikit-learn.org. 2020. 3.1. Cross-Validation: Evaluating Estimator Performance — Scikit-Learn 0.23.2 Documentation. [online] Available at: <https://scikit-learn.org/stable/modules/cross_validation.html> [Accessed August 2020].

[15] Scikit-learn.org. 2020. Sklearn.Model_Selection.Cross_Val_Score — Scikit-Learn 0.23.2 Documentation. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html> [Accessed August 2020].