Kaggle Competition Report

Toxic Comments Classification

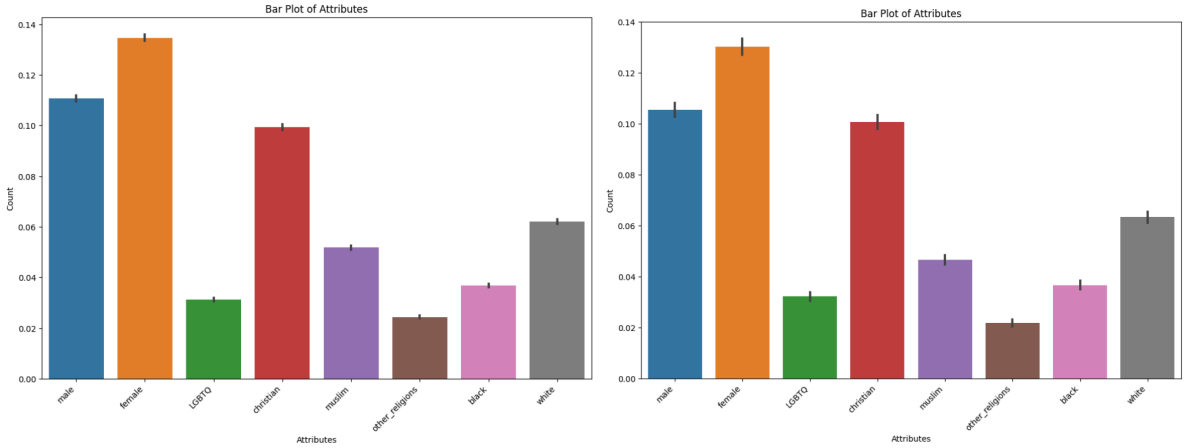
Hongyang YE, Peter KESZTHELYI

Abstract

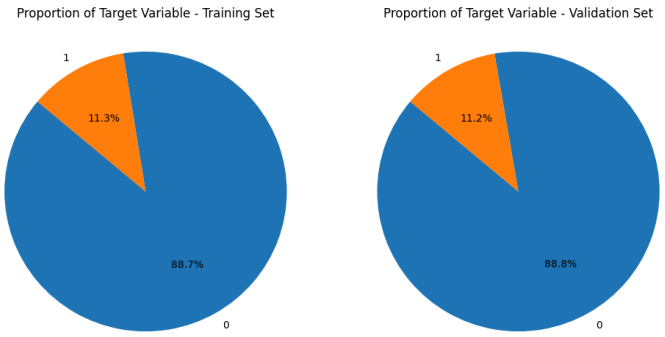
In this report, we present our approach and methodologies employed in the Kaggle competition for Toxic Comments Classification. First, we present our insights from exploratory data analysis. Next, we explain our experimentations with data preprocessing, embedding techniques, and model architectures, such as LSTM, GRU, RNN, and CNN. The report also discusses challenges, findings, and potential avenues for future development.

Exploratory Data Analysis

We conducted exploratory data analysis to understand our data better. The distributions of each group in both training dataset and validation dataset are presented below.



According to the bar plots, it is clear that our datasets are both imbalanced. Classes of male, female, and christian have the largest amount of samples, while other_religions, black, and LGBTQ have relatively limited samples. This can affect our accuracy crucially. How we tried to tackle this problem will be discussed in detail in later sections.



Furthermore, we note that our target variable is also unbalanced, with just roughly 11% of comments being toxic, while the rest is clean.

Data Preprocessing

Since the data of this kaggle competition are toxic comments, it is important to preprocess the data so that our models would be able to learn from the correct patterns and then label the comments.

We first created a dictionary used for replacing the letters and digits in the wrong format. Then, we applied regex to remove all non-alpha numeric and space. We also clean some words that might not be useful for model's learning, like the url link of some websites. We also noticed that there are unicodes, emojis, and tones in the comments, and they were also cleaned.

Furthermore, we also dropped the images, css, cleaned the templates, and replaced some patterns and punctuations in wrong format. With close observation to the data, we cleaned some specific patterns to help our models learn from the comments. For example, there are a lot of typos or casual words for lgbt, like lbgt, bglt, lgtb, lgtbbbbbbb, mslgbt, etc. We corrected the typos to make sure our models can learn from the correct patterns.

We checked our data cleaning process with embedding. Without data preprocessing, the amount of the null words cannot be embedded is 24831, and it decreased to 19548 after data cleaning, proving that our data preprocessing is effective.

Embedding

Before training different models, we also need to build the embedding matrix. We utilised the tokenizer from Keras framework, and tokenized the cleaned training dataset and validation dataset, and created an embedding matrix for words in the dataset using pre-trained word embeddings. We tried with two pre-trained word vectors, crawl-300d-2M.vec, glove.840B.300d. The table below presents the results with different embedding matrices:

WordVectors	**#wordvectors**	**#unique tokens**	**#null word**	**WGA**
Crawl	2,000,000	190437	29138	0.7585
Glove	2,200,000	190437	28316	0.7730
Crawl+Glove	2,840,000	190437	27155	0.7687

The results above were gained with the same model, a relatively basic recurrent neural network, and with the same amount of epochs. From the results, the glove word vectors had the best performance, and the combination of two pre-trained word vectors didn't perform better. As a result, we would use only the glove word vectors for embedding layers in the following sections, except the BERT model.

Numbers of Epochs

Due to the complexity of our models, it seems that they can overfit very easily after a few epochs of training. We tried to find the number of epochs from where our model start to overfit, and the results are shown in the table:

#Epochs	1	2	3	4	5

MLP	0.6858	0.6834	0.6889	0.6858	0.6870
RNN	0.7608	0.7603	0.7590	0,7712	0.7657

Apparently, the number of epochs of which the model starts to overfit differs from models. Besides, stacking all of our models also requires diversity among the models. Therefore, in the following section, the number of epochs for training each model also varies.

Cross-Validation

We also applied cross-validation to obtain more stable results and avoid overfitting. Similarly, in consideration of diversity among the models, we apply different numbers of folds for cross validation on different models.

Models

LSTM

Apart from the baseline model, the first model we tried in this competition is the LSTM model, and the architecture of our LSTM model is shown as follows:

Layer (type)	**Output Shape**
InputLayer	[(None, 350)]
Embedding	(None, 350, 300)
Spatial Dropout 1D	(None, 350, 300)
Bidirectional	(None, 350, 100)
Dropout	(None, 350, 100)
Bidirectional	(None, 350, 100)
Dropout	(None, 350, 100)
GlobalAveragePooling1D	(None, 100)
GlobalMaxPooling1D	(None, 100)
Concatenate	(None, 200)
Dense(Relu)	(None, 50)
Dense(Sigmoid)	(None, 1)

In our LSTM model, we have two Bidirectional LSTM layers to process the input sequence both forward and backward, capturing information from both directions. Then we applied two global pooling layers to the output of the bidirectional LSTM layers to capture different aspects of the information in the sequence. Besides, we continuously added dropout layers between the functional layers to avoid overfitting.

GRU

Considering our limited data for some subgroups, shown in the section of exploratory data analysis, we also tried the GRU model, which is comparatively simpler than the LSTM model and may be more suitable when dealing with smaller datasets.
The overall architecture of our GRU model is similar to the LSTM model, with the two Bidirectional LSTM layers replaced by two Bidirectional GRU layers.

Attention

One limitation of RNN models is that it is sequence to sequence, and may miss information of word's positions in one sentence. Therefore, we also introduced the attention layer into both our LSTM and GRU models. We believe that the attention layer can help our RNN models better capture long-range dependencies, and then improve our models' performance and robustness.

CNN

With the purpose of stacking different models and obtaining a better result with a second-order model, we need to have models other than recurrent neural networks. Therefore, we also tried the textCNN model and added it as one of the first-order models when stacking. The following table presents the architecture of our textCNN model.

Layer (type)	**Output Shape**
InputLyaer	[(None, 350)]
Embedding	(None, 350, 300)
SpatialDropout1D	(None, 350, 300)
Reshape	(None, 350, 300, 1)
Conv2D	(None, 350, 1, 32)
Conv2D	(None, 349, 1, 32)
Conv2D	(None, 348 1, 32)
Conv2D	(None, 346, 1, 32)
MaxPooling2D	(None, 1, 1, 32)
MaxPooling2D	(None, 1, 1, 32)
MaxPooling2D	(None, 1, 1, 32)
MaxPooling2D	(None, 1, 1, 32)
Concatenate	(None, 4, 1, 32)
Flatten	(None, 128)
Dropout	(None, 128)
Dense	(None, 1)

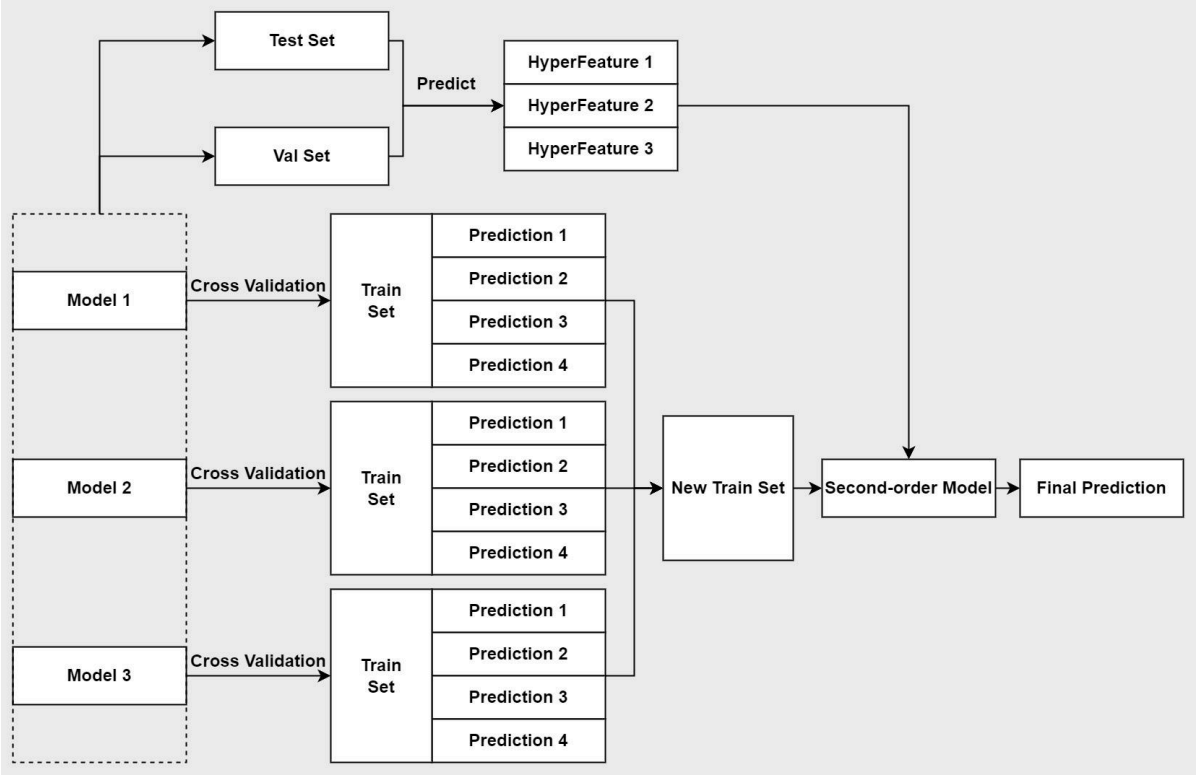
In our textCNN model, we have four convolutional layers with different kernel sizes to capture different n-gram features from the input. Four MaxPooling layers are then applied to the output of each convolutional layer to extract the most important features. In the context of this competition, it is possible that the textCNN model is better than RNN model because textCNN model excels at capturing local patterns and relationships within short sequences of text. Our data, according to the result of EDA, are comparatively short sentences.

BERT

We also experimented with creating fixed-size sentence embeddings using a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model, and using these vector embeddings as the input for our various neural network models. We were hoping that BERT embeddings would capture the semantic meaning and context of the text. To that end we encoded the tokenized text into 768 size vector representations.

Stacking

At the end of this competition, we tried stacking our models altogether to see if it can have better performance. First, we conduct cross validation for each model, and concatenate the predictions of each validation set to be the predictions of the entire train set. If we have 3 models to stack, the predictions of the train set would be 3 new features as our new train set. Meanwhile, we make predictions on both test set and validation set to obtain hyper features which would be later used for predictions with the second-order model.
We then train our second-order model with this new train set. We tried Linear Regression, Logistic Regression, and SVM as our second-order model. Finally, we import hyper features obtained from the base models to the second-order model and predict the ultimate results of our comments. How we stack our models is shown as follows.



Results

Below we present the results for select models from the ones that outperformed the baseline model of the competition. The metric is the worst group accuracy computed on the validation set.

Model	**WGA (Validation set)**
LSTM	0.7889
Stacking (LSTM, GRU, GRU with attention)	0.7742
GRU (with attention layer)	0.7723
GRU	0.7615
BERT Embedding, MLP (+1 hidden layer)	0.7210
BERT Embeddings, MLP (baseline)	0.7107
CountVectorizer, MLP (baseline)	0.7096

Conclusion

In this Kaggle competition, our approach to toxic comments classification involved extensive experimentation with various preprocessing techniques and transformations, namely, word to vector embeddings (Crawl, Glove) and sentence embeddings using a transformer architecture (BERT), following in-depth data cleaning. Using different representations of our original data, we implemented a wide range of model architectures, including simple neural networks, LSTM, RNN, GRU, and CNN models.

Our results were competitive in performance on the Kaggle competition leaderboard, indicating that our model was able to learn the aspects of toxic comments that we intended to capture. The best performing models relied on pre-trained word embeddings and made use of the sequential nature of the textual data (LSTM, GRU). Although using BERT embeddings (instead of CountVectorizer) improved the performance of the baseline model, the results using more complex models were subpar.

In terms of further developments, data augmentation is a promising approach to reduce the imbalanced nature of our training set. It is possible to generate comments in specific categories, using machine translation to a different language and then back, producing a slightly altered phrasing of the same comment as the original. This would pave the way to having access to a larger dataset with better characteristics and a more balanced distribution across categories.