Group Project - Toxic Comment Classification

Team members: Piangpim CHANCHARUNEE, Ruxi HE, I-Hsun LU

Introduction

The goal of our Kaggle competition was to develop a toxic comment classifier. Fixing biases in these classifiers is important in order to maintain a safe and fair environment online. This competition in particular focuses on the subpopulation shift problem. Different demographic groups represent these subpopulations, and the goal is to create a model that performs well for all groups and reduce biases in comments about certain groups. The key challenges of the competition include dealing with the imbalanced labels, as toxic comments and group mentions are not evenly distributed. Apart from the baseline MLP model, we used ResNet with Attention Mechanism, paired with FastText embedding to significantly reduce the training time to 3 mins per epoch at the cost of a mere 0.02 accuracy drop. Then, to further boost the performance, we opted for the BERT and eventually yielded a 0.78[1] accuracy, 0.08 up from the baseline model.

Dataset and Problem Analysis

First, we observed the data structure. The dataset consists of 269,038 training samples. The target variable is imbalanced, with significantly more non-toxic comments than toxic ones, which could affect model performance and fairness. Although the number of samples is equal across all demographics, one sample could belong to multiple demographics, such as “black” or “LGBTQ”. This overlap can cause certain combinations of demographics to be underrepresented or overrepresented, potentially leading to biases in the model. Additionally, the way the demographics appear in the test set may differ from the training set. If certain groups have more mixed identities in training but appear separately in testing, the model might misclassify those samples, leading to a drop in performance when applied to a new demographic distribution.

Experiments

Method #0: CountVectorizer + 1-hidden-layer MLP

In baseline.ipynb provided, the dataset uses CountVectorizer to preprocess the data, without any data cleaning beforehand; the model is a rudimentary 1-hidden-layer MLP. The training time takes around 20 mins in the Kaggle basic CPU environment per epoch and strikes around 0.70 accuracy.

Method #1: FastText Embedding + Residual Neural Network with Attention Mechanism + weighted loss based on demographics accuracies

[1] Every accuracy in this report is rounded to the hundredth decimal.

A lightweight model is preferred at first hand given the time and resources constraints. 20-min-per-epoch training time still has room for improvement. The huge dimensions of the output by the CountVectorizer are the main cause for such a long training time. So we adopt stopword removal as data cleaning techniques and FastText embedding preprocessing to reduce the dimension from hundreds of thousands to 300. We prefer FastText over GloVe for its outstanding out-of-vocabulary performance which is effective in non-standard User-Generated-Content dataset.

To enhance MLP model performance, we introduce the attention mechanism to capture relationships between words in a sentence or document , residual module to improve gradient flow, and batch normalisation layer to stabilize training and improve convergence. Given that label 0 extremely outnumbers label1, we assign samples labelled with 0 more weight when calculating the loss. We adjust the loss further according to the performances among different demographics from the last epoch, with a focus on the demographics in which model performances are worse, in an attempt to address the subpopulation shift problem.

The training time is significantly reduced to around 3 mins per epoch with a 0.69 worst group accuracy according to the leaderboard. But the problem encountered is that the accuracy reaches the plateau after the 1st epoch. We conclude that it is due to the model limits after attempts of adjusting batch size, hidden layer dimension, learning rate, adding scheduler, etc.

Method #2: DistilBERT, Adversarial Debiasing, and DRO

In the second method, we focused on upgrading the model to a more powerful one while addressing the subpopulation shift problem. Therefore, we decided to add two main features. The first feature we implemented to address imbalance was adversarial debiasing, with a Gradient Reversal Layer to remove demographic features from toxicity predictions. The next feature we implemented was Distributionally Robust Optimization (DRO), to ensure the model performs well across worst-case subpopulations. We did this by assigning weights based on group_membership.

However, the codes that we used for these two features did not directly address the fact that the groups may overlap. Therefore, it was not as effective and was not utilized to its full potential. In the future, if we were to further improve this model, we would change the codes to account for this fact. For example, we should have used BCEWithLogitsLoss instead of CrossEntropyLoss in order for the adversary to be able to classify a sample into multiple groups rather than just one. In regards to the DRO, the group loss calculation should have been modified by normalizing by the number of groups a sample belongs to rather than taking the sum over all samples in a group. Weight clipping should also have been implemented to prevent any single group from dominating.

In terms of the model, we ran the baseline model combined with the previously mentioned features. However, we wanted the model to be more accurate. We had previously used BERT (see method 3) and it provided us with accurate results; however, the model was very computationally demanding and took hours to run. Therefore, in order to improve accuracy

but reduce training time, we picked DistilBERT, which trades a bit of accuracy for faster training time. We tried a number of combinations between the baseline model, DistilBERT, adversarial debiasing, and dataset reweighting. We trained most of the models for one epoch. Training times were around 40 - 60 minutes per epoch, with a final accuracy of 0.78.

Method #3: BERT, large BERT

To address the limitation of the training dataset size of the comment dataset we have, we chose to adopt a pre-trained model for our specific task of identifying toxic comments. With pre-trained models like BERT, we can benefit from the massive dataset the creators used for training them, not only being restricted from the dataset we have now for the project. Pre-trained models are self-supervised, meaning they do not require manual annotation of the training data. This characteristic allows the training data to be massive, helping overcome the bottleneck of human-labeled data, enabling the model to be trained on larger datasets provided by companies or research laboratories with more resources. Other users, like us, can then use the pre-trained model as a foundation to develop models tailored to more specific tasks.

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model by Google. It uses Masked Language Modeling (MLM) to process text bidirectionally and Next Sentence Prediction (NSP) to understand sentence relationships, enabling effective performance in tasks like text classification and question answering..

We first used BERT with a subset of the training data and achieved promising results, with a score of 0.78 after 1 epoch and 0.78 after 3 epochs. We also experimented with BERT Large to evaluate whether the size of the pre-trained model significantly impacts the results. The results showed that BERT Large achieved a score of 0.77 after 1 epoch of training, which is comparable to the performance of the base BERT model. Theoretically, the performance of models with more training data or larger pre-trained model should be better, but this is not the case. The suboptimal performance is likely due to overfitting, where the model memorizes training data instead of generalizing. BERT Large performs worse because it requires more data to fully utilize its capacity, making it prone to overfitting on smaller datasets. Additionally, computational challenges or lack of proper learning rate scheduling, may also have hindered its performance.

Results and Method Comparison

Method#	0	1	2	3
Data Preprocess	CountVectoriser	Stopword Removal, FastText Embedding	DistilBERT Tokenizer	BERT Tokenizer
Model	1-hidden-layer MLP	Attention ResNet	DistilBERT	BertForSequence Classification

Loss function	Binary Cross-Entropy Loss	Binary Cross-Entropy LossWithLogitsLoss	Binary Cross-Entropy Loss	Binary Cross-Entropy Loss
Optimizer	AdamW	AdamW	AdamW	AdamW
Learning rate	0.05	0.05	2e-5	2e-5
Training time	~20’ per epoch	**~3’ per epoch**	~40 - 50’ per epoch	~3 per epoch
Training environment	Intel Xeon 2.20 GHz CPU (Kaggle CPU)	Intel Xeon 2.20 GHz CPU (Kaggle CPU)	Google Colab T4 GPU	Nvidia T4 GPU
Best WGA at test set	0.71	0.69	**0.78**	**0.78**

Each method improved performance while balancing accuracy, fairness, and efficiency. Method 0 was a simple baseline with slow training and moderate accuracy. Method 1 improved efficiency by reducing input dimensions, but accuracy dropped slightly. Method 2 brought a major boost in performance, leveraging pre-trained embeddings and adversarial debiasing, but required longer training. Finally, Method 3 achieved similar results but had higher computational costs and risk of overfitting. Overall, BERT-based models performed best, but efficiency and bias handling remain areas for improvement.

Conclusion And Possible Applications Scenarios

In this competition, we explored different models to improve toxic comment classification while addressing bias and subpopulation shift. First, we tried a simple MLP and FastText embeddings for faster training. Then, we used DistilBERT with adversarial debiasing and DRO to make the model fairer, but it did not fully account for overlapping demographic groups. Finally, we experimented with BERT and BERT Large, achieving our best accuracy of 0.78 after one epoch. Each model we implemented has distinct trade-offs in terms of accuracy, fairness, computational efficiency, and interpretability. Future improvements should focus on better bias correction techniques and efficient fine-tuning to enhance both fairness and accuracy.

Depending on the practical application, different models may be preferable. Method#1 is suitable for low-resource and real-time moderation scenarios e.g. on-device content filtering for mobile applications where computational power is limited. Method#2 is best suited for fairness-sensitive applications in large-scale moderation such as government forums, healthcare discussion boards where bias mitigation is crucial. The Method#3 is best suited for high-accuracy applications in large-scale, well-funded platforms like YouTube, Twitter, Facebook where accuracy is a top priority and computational resources are available.