Toxic Speech Classification with Fine-tuned BERT

Tongkai Yang
tongkaiy@usc.edu

Shihan Xu
shihanxu@usc.edu

Chengyuan Zhou
czhou69@usc.edu

Ruichao Ma
ruichaom@usc.edu

Jiawen Song
jiawenso@usc.edu

Abstract

This paper introduced the method to classify English toxic comments using state of art transformers. We fine tuned the BERT model and evaluated it using Kaggle's toxic comment classification dataset and twitter insult comment dataset. Also, we use pseudo labeling technique on testing data to enlarge our training dataset, which boosted the performance of our model. We achieved a very good performance with accuracy 98.9% for binary classification (toxic vs. non-toxic). Moreover, by combining this classifier with a speech recognition system, we created a high quality toxic speech classifier which may be used practically to detect toxic talk or conversation.

1 Introduction

Toxic content recognition has always been a popular task in NLP field. Decades ago, people have created toxic word filter which can filter out bad words. In recent years, with appearance of more and more advanced language models, toxic content classification in sentence level is researched by lots of NLP experts. This project aims to use pre-trained transformer model to identify toxic comments. Moreover, we decide to incorporate toxic text classifier with a speech recognition system. Thus, we deliver a high quality toxic speech classifier which is able to detect speech that hurt people and influence emotion negatively. We believe that this project can provide some insights and extra techniques on text classification.

2 Related Work

Before we start this project, we browse Github and Kaggle to see how other people do similar projects[1, 2, 7, 6, 8]. One good resource is a project called Detoxify[1]. They mainly use bert-base-uncased and xlm-roberta-base transformer to detect toxic comments and classify them in multilingual environments. They also gave scores of toxic levels for each comment, with very high accuracy around 0.93. However, they still have their own limitations. They did not do data preprocessing and cleaning. Moreover, their dataset used in training is small. We believe doing data preprocessing and train on a larger dataset will help improve model performance. That's what we will do later in our project. We also do some research on Kaggle competition for this topic[2]. Among top teams, their ROC AUC scores around 0.988, which is very high. Their general steps include adding smaller models trained with Albert for original and unbiased models, and updated unbiased model weights used by Detoxify. However, even though top teams did really well on prediction, their problems are still common: ignore data preprocessing and their data amounts are small. In this project, we make some improvements on the amount of training data and data processing.

3 Software and Hardware

This project is done by Python3. The model architecture is implemented by PyTorch. Pre-trained model (BERT) is from HuggingFaces's transformer pool[3]. To obtain the as much computation resource as possible, we use Colab Notebook with GPU accelerator.

[1]https://github.com/unitaryai/detoxify
[2]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/leaderboard
[3]https://huggingface.co/bert-base-uncased

4 Datasets

We use the jigsaw toxic comments dataset as our main dataset [4]. This dataset contains 223549 comment texts crawling from Wikipedia comments. Two extra data sources come from Kaggle's competition – Detecting Insults in Social Commentary[5] and Twitter's offensive language detection project[6]. All labels in these datasets are labeled based on text's toxicity by human raters. Besides toxic or non-toxic label, labels in other categories with smaller granularity are also provided in these datasets. However, since this project is dealing with a binary classification task, we only consider binary labels (toxic vs. non-toxic).

5 Data Preprocessing

Data preprocessing is completed using Python and some text process libraries such as NLTK and Spacy. We first remove user id and IP address, URL and HTML format using regular expression because this information does not provide any information related to language toxicity. Then punctuations, special tokens and extra white spaces and lines are removed. We also transform all accented characters into unicode version. For instance, we transform café to cafe.

Some extra important steps in data processing are to remove stop word, word stemming and lemmatization[5]. Stop words contain no information about text sentiment. And word stemmer helps remove suffixes and convert, for example, "mourning" to "mourn". Therefore, we use the NLTK's PorterStemmer module to conduct the process. Also, lemmatization performs normalization using vocabulary and morphological analysis of words. Lemmatizations aim to remove inflectional endings only and return the base of the dictionary form of a word. So we perform it.

Moreover, to deal with word contraction, we establish a big contraction dictionary which constitutes 80 different kinds of shortened words, such as *could've* to *could have*, *hasn't* to *has not*, *he'd've* to *he would have* and etc.

In the process of data processing, we found that some words are misspelled. Among them, some words are originally misspelled and some words are misspelled after some data cleaning steps. Misspelled words cause trouble for classification tasks in our projects. To solve this, we use PyspellChecker[7] to standardize spelling. It works by finding all permutations of characters in a word with predetermined edit distance. It finds all strings that can be created by inserting, deleting, replacing and transposing characters in a given word. It compares each permutation to a dictionary of known words and their frequencies and returns words with highest frequency as the most probable correct spelling. By using this package, we successfully correct 78% of misspelled words.

Since our dataset includes twitter comments, which is one main social media platform. There is an explosion in the usage of emojis in our day to day life as well. We need to remove emojis but keep emotions, since emotions give some valuable information for sentiment analysis. One good way to keep emotion in our project is to convert emotions into word format so they can be used in downstream modeling processes. Specifically, we found a comprehensive list of emojis with corresponding words from one Github repo[8] and this list fits our project well.

After all above process, we merge all three training datasets into one. Finally, the training dataset has approximately 250000 records, with ratio of "non-toxic" vs "toxic" as 4:1.

6 Methodology

6.1 Model Selection

The model used for this classification task is BERT. Previous experiments have shown that transformers had very good performance on a variety of NLP tasks. Due to the way BERT is trained, it has shown to be quite effective on English text classification. We also added a fully connected layer on top of BERT output as our classifier layer. And our output layer is a sigmoid layer which can turn model output into probabilities within range [0, 1]. To make this model well suited to our project task, we decided to fine-tune BERT along with the last classifier layer. (insert model visualization here)

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

[5]`https://www.kaggle.com/c/detecting-insults-in-social-commentary/data?select=train.csv`

[6]`https://github.com/t-davidson/hate-speech-and-offensive-language`

[7]`https://github.com/barrust/pyspellchecker`

[8]`https://www.kaggle.com/eliasdabbas/how-to-create-a-python-regex-to-extract-emoji/data`

6.2 Data Encoding and Padding

We used BertTokenizer from a Transformers library to encode the data. In the project related work description we found on Github, they used the model “bert-base-uncased.” However, since the case is a crucial factor to determine whether a sentence is friendly or not, we went with the “bert-base-cased” as the pretrained model.

We typically use the length of the longest sentence as the max-length for the padding. However, since some sentences can be too long for the dataset, we padded sentences in batches. Additionally, we chose the length of the longest sentence in the batch as the max-length for this single batch.

6.3 Semi-supervised Data Augmentation (Pseudo labeling)

As we mentioned in section 4, there are only about 250000 training data while there are more than 150000 testing data. To get the best result of classification, it is obvious that we are lack of training data. Therefore, we think of a semi-supervised way which is pseudo labeling to enlarge the training data pool. Pseudo labeling is proposed by Lee[3] in 2013. There are 5 core steps:

1. Train a model on limited labeled data.
2. Use this data to predict large amounts of unlabeled data.
3. Use predicted labels to compute unlabeled loss.
4. Add labeled loss with unlabeled loss using different weights.
5. Backpropagate to minimize the loss.

One interesting part of pseudo labeling is that instead of simply adding unlabeled loss with labeled loss during training, it uses weighted loss as the final loss function.

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} Loss(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} Loss(y_i'^m, f_i'^m) \tag{1}$$

The weights are calculated as formula (2). It helps the model to add more information from unlabeled data.

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1 \le t < T_2 \\ \alpha_f & T_2 \le t \end{cases} \tag{2}$$

7 Experiments

7.1 BERT Baseline

BERT is an extremely large pretrained model with 12 layers and 110 million parameters. We use an bert-base-uncased transformer model written by the Hugginface team and incorporate it with a PyTorch linear layer. To avoid over-fitting and gradient exploding, we used AdamW which is basically Adam optimizer with weight decay equal to 0.1[4]. Furthermore, advanced learning rate scheduler is used to adjust learning rate during training process. After experiments, we decide to use linear scheduler with warm up which means that the learning rate will be warmed up from 0 to preset value and then linearly decrease in n steps where $n =$ number of batch $*$ number of epochs $-$ warmup steps.

In terms of loss function, binary cross entropy loss is used to deal with the binary classification task. One thing to note is that sigmoid activation function does not come with binary cross entropy. So it is necessary to add a sigmoid layer at the end of our model structure.

Due to the computation resource and time limit, the baseline model is trained in 3 epochs. After each epoch, we evaluate the performance on validation data. The evaluation metrics are discussed in model evaluation section.

7.2 BERT with Pseudo Labeling

The semi-supervised method provides us with more training data. Based on the formula of pseudo labeling weight, we set T1 = 100 and T2 = 500, initial step = 100 and alpha = 3. We repeat the pseudo labeling process for 5 epochs. With each epoch, the model is trained on all training data as every 100 batches of unlabeled data get pseudo labels. At the same time, step increases by 100 so that the weight for unlabeled data will increase. In summary, chart xxx explains how we implement pseudo labeling.

8 Speech to Text Model

We choose the IBM Watson Speech to Text service because this service provides APIs that use IBM’s speech-recognition capabilities to produce transcripts of spoken audio. Since our project is mainly focused on English audio transcription and toxic word detection, the speech-to-text model is also using an pre-trained English model called ’en-

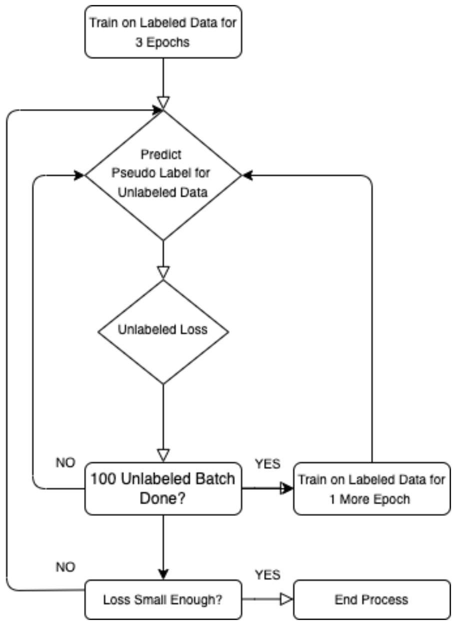
Train on Labeled Data for 3 Epochs
Predict Pseudo Label for Unlabeled Data
Unlabeled Loss
NO
100 Unlabeled Batch Done?
YES
Train on Labeled Data for 1 More Epoch
NO
Loss Small Enough?
YES
End Process

Figure 1: Pseudo-labeling Process

US_BroadbandModel'[9]. The model also supports multiple different audio formats such as .mp3 and .flac. The recognized audio transcript will be in string format. Then the result is fed to the above-mentioned toxic word classification model to actually classify the result.

9 Results

Model	BERT Baseline	BERT w/ Pseudo-Labeling
Accuracy	0.942	0.975
Precision	0.883	0.92
Recall	0.866	0.901
F1	0.883	0.91

10 Discussion

Prior works have achieved excellent performance on toxic classification task. Our base line BERT model has similar performance as those works with F1 score equals 0.883. However, due to the fact that we obtain more training data sources and use supervised learning method to augment data, the performance of BERT model with pseudo labeling is boosted dramatically. All evaluation metrics increase by around 0.3. This means that this model is able to accurately detect most of toxic or offensive language.

[9] `https://cloud.ibm.com/apidocs/speech-to-text?code=python`

11 Future Work

The limitation on computation resources, especially GPU memory, prevents us from training the model for more epochs. In the future, we plan to achieve better performance by giving more training epochs and repeating pseudo labeling more times. Furthermore, we can choose different language model such as LSTM and GPT-2 to do more experiments. And this project can be extended to multilingual toxic comment classification.

References

[1] Mai Ibrahim, Marwan Torki, and Nagwa El-Makky. "Imbalanced toxic comments classification using data augmentation and deep learning". In: *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2018, pp. 875–878.

[2] Harsh Kajla, Jatin Hooda, Gajanand Saini, et al. "Classification of online toxic comments using machine learning algorithms". In: *2020 4th international conference on intelligent computing and control systems (ICICCS)*. IEEE. 2020, pp. 1119–1123.

[3] Dong-Hyun Lee et al. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. 2013, p. 896.

[4] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[5] Fahim Mohammad. "Is preprocessing of text really worth your time for online comment classification?" In: *arXiv preprint arXiv:1806.02908* (2018).

[6] Mujahed A Saif et al. "Classification of online toxic comments using the logistic regression and neural networks models". In: *AIP conference proceedings*. Vol. 2048. 1. AIP Publishing LLC. 2018, p. 060011.

[7] Darshin Kalpesh Shah et al. "Multilabel Toxic Comment Classification Using Supervised Machine Learning Algorithms". In: *Machine Learning for Predictive Analysis*. Springer, 2021, pp. 23–32.

[8] Sara Zaheri, Jeff Leath, and David Stroud. "Toxic comment classification". In: *SMU Data Science Review* 3.1 (2020), p. 13.