Natural Language Processing Project

Toxic Comment Classification

Grosjean Guillaume, Chan Rémi, Duzenli Mikail

April 26, 2022

Contents

1 Introduction

This project consists in building a model able to detect toxic comments. It was originally a Kaggle competition from 2017. It was submitted by the Conversation AI team, a research initiative founded by Jigsaw and Google working on tools to help improving online conversation. More specifically, the goal is to detect different types of toxicity like threats, obscenity or insults in online comments.

After a short presentation of the dataset, we introduce a first simple model to tackle the task. Since this Kaggle competition ended in 2017, best models were based on LSTM, GRU and encoder/decoder structure. In the last part of the report, we therefore introduce a Transformer based model and compare it to best 2017 models.

2 Data

2.1 Training data

The training dataset provided in this challenge is a large number of Wikipedia comments in English which have been labeled by human raters for toxic behavior. It can be accessed via this link. There is a total of **159,571** training examples, each of them consisting of the comment itself and a one-hot representation of 6 different toxic comment classes. Note that this is a multi-class classification task, where a single example can belong to zero, one or several classes. The different classes and the percentage of examples belonging to each class is shown figure 1

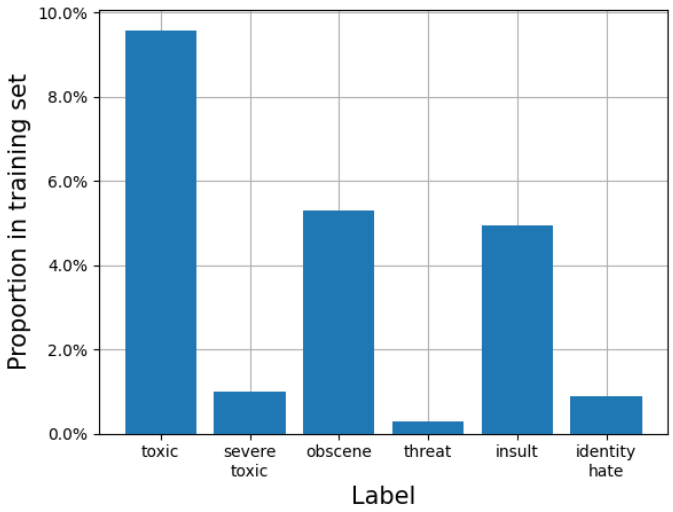

Figure 1: Percentage of data belonging to each class among the training dataset. Most of the data is non-toxic.

2.2 Test dataset

The test dataset is provided with 153,164 examples that follow the same structure as the training data. Among all examples, some were not used to test models in the official competition and are assigned with a -1 label for every class. After removing those examples, we end up with a total of **63,978** test examples. Percentage of examples belonging to each class is shown figure 2.

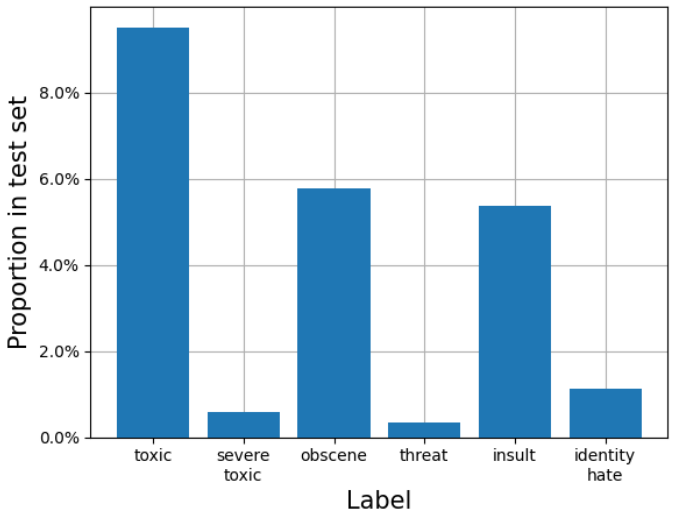

Figure 2: Percentage of data belonging to each class among the test dataset. The test data distribution is similar to the training one.

An overview of the most common words in each class can be seen on fig 3.

Figure 3: Word clouds of the different classes

2.3 Data cleaning

As training and test comments come from real Wikipedia comments, the text contains a lot of noise. We thus have to clean and normalize the data. We remove URLs, punctuation, digits, multiple spaces, special characters... (see the notebooks for the full cleaning function). As it is a classification task, we don't need perfectly constructed sentences thus we can apply lemmatization and remove stop words from the comments. We use NLTK library [2] to apply such transformations. An example of every step of data pre-processing is shown table 1.

	Comment text
Original	Citation needed Citation needed - but I cannot work out how to do it. http://www2.anyoneforpimms.com/winterpages/winter_history.html
Cleaning (noise removal)	citation needed citation needed but i cannot work out how to do it
Cleaning + Lemmatization + Stop words removal	citation needed citation needed cannot work

Table 1: Example of every step of data pre-processing on a Wikipedia comment.

We also remove non-English comments with the *fastText* Python library. This enables to clean the very few occurrences of other languages *e.g.* French (310 occurrences) or Polish (39 occurrences).

3 Models

3.1 Task and metrics

In this project, we are addressing a multi-class classification task. For a given comment, we aim to predict whether it belongs to each of the 6 toxicity classes. The output will be a 6-dimensional vector with each value being the probability of a given comment to belong to each toxicity class.

We will evaluate our models with the metric used in the Kaggle competition: the mean class-wise ROC AUC. In other words, the score of a model is the average of the individual AUCs of each predicted class.

3.2 Bag of words and simple classification model

The second step is to make a first simple classification model as a base result we will refer to later. We build two different types of bag of words : binary and *TF-IDF* (term frequency-inverse document frequency). To do so, we need to build the vocabulary based on the training dataset by counting the occurrence of each encountered word and only keep those that are more frequent than a defined threshold. By setting the threshold to 50 occurrences, we have a vocabulary of 4,266 words versus 102,991 before thresholding. We thus manage to keep words that will be relevant for the model but we are impacted by the spammers. We build the binary dataset by marking the presence of words as a boolean value, 0 for absent, 1 for present. The *TF-IDF* dataset is built using the *sklearn* Python library.

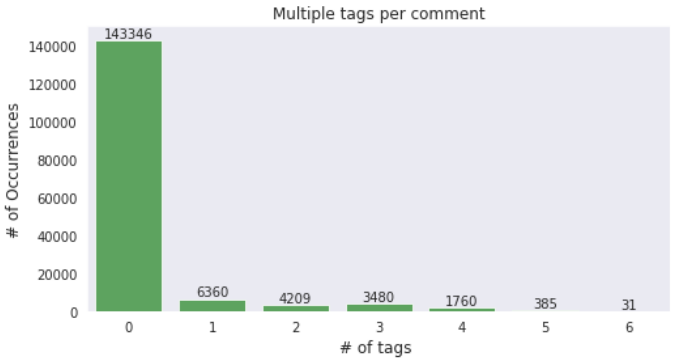

Figure 4: Histogram of class occurrences

Before going through the training, we must solve the problem of imbalanced data. Indeed, we can see in figure 4 that the number of "clean" comments is more than 10 times superior than the number of toxic comments belonging to 1 or more classes. This would lead to many difficulties in the training such as leading the model to always classify data as non-toxic. To solve this problem, we randomly drop 100,000 "clean" comments from the dataset. This approaches solve both the problem of imbalanced data and the problem of memory, as the entire train dataset weights 7 GB of data.

For the training, we use a simple fully connected sequential model with dense layers and dropout. We use batch normalization and ReLU non-linearities between all the hidden layers. Because of the simplicity of the bag of words approach, the model quickly overfits after 3 epochs so we only train for that long.

Epoch	Mean ROC/AUC for binary vectors	Mean ROC/AUC for tf-dif vectors
1	0.9329	0.9624
2	0.9475	0.9663
3	0.9487	0.9611

Table 2: Mean ROC/AUC computed on validation data during training

On the test set, we have the following results :

Mean ROC/AUC for binary vectors	Mean ROC/AUC for tf-dif vectors
0.952	0.966

Table 3: Mean ROC/AUC computed on test data

We can see that, as expected, the tf-idf vectorization leads to slightly better results.

3.3 Transformer model

Today's state of the art results on text classification tasks are achieved with transformer models. BERT [4] models, from Google AI Language, can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks [5]. We will thus fine-tune a small BERT model for our specific classification problem. This can be done with the help of the HuggingFace Transformers library [7].

The data pre-processing is similar to section 2.3, except that we don't use lemmatization neither stop words removal. The reason is that BERT is a contextual model, so we need to keep as much information as we can about the context of each word.

Because of our limited computational resources, we use a small version of pre-trained BERT, named BERT-small [1] [6]. It has $L = 4$ Transformer layers and $H = 512$ dimensional hidden layer, for a total

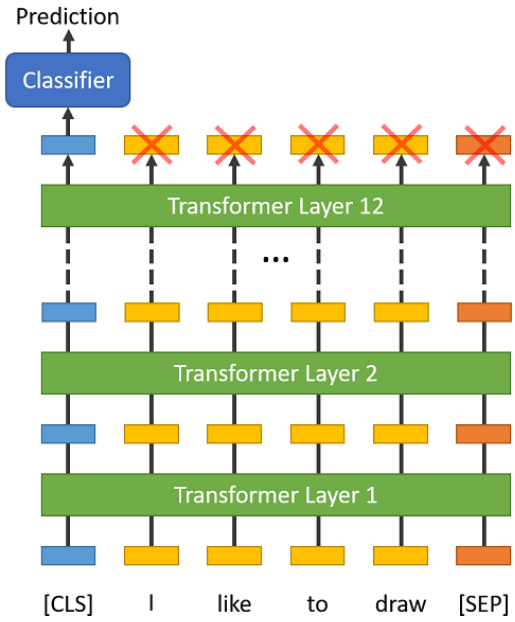

Figure 5: When fine-tuning BERT for a classification task, only the [CLS] is used by the the classifier (image taken from [3]).

number of 28M parameters. We use the pre-trained BERT tokenizer *bert-base-uncased* to tokenize our cleaned data, with a maximal length of 512 (longer comments are truncated). To use BERT for classification, we add an additional fully-connected layer taking as input the [CLS] token of BERT output (512-dimensional) that is, by design, supposed to summarize all the information needed for classification (see figure 5). The output dimension of the fully-connected layer is 6, corresponding to the number of classes, with a sigmoïd activation. When training, the entire pre-trained BERT-small model and the additional untrained classification layer is trained on our specific task.

In the original BERT paper [4], the authors recommend choosing from the following values to fine-tune BERT for a specific task:

- **Batch size**: 16, 32
- **Learning rate (Adam)**: 5e-5, 3e-5, 2e-5
- **Number of epochs**: 2, 3, 4

We chose a batch size of 32, a learning rate value of 2e-5 and 3 epochs. We use Adam as optimizer and binary cross-entropy as loss function. We save the model after each of the 3 epochs and show the performance of the 3 saved models in table 4. The metric is the average ROC/AUC scores of each of the 6 classes.

Epoch	Mean ROC/AUC
1	0.9832
2	0.9845
3	0.9822

Table 4: Mean ROC/AUC score computed on the test dataset using model saved at epoch 1,2 or 3.

After two epochs, the model starts overfitting. We get the best metric score at epoch 2. As expected, we get a better score than with the bag of word model, without having to deal with the imbalanced data.

However, we do not beat the top score of the Kaggle competition, which was reached by the team Toxic Crusaders with a mean AUC/ROC score of 0.9885.

There are two main ways we can think of to increase the performance of our model, that both require more computational resources. On the one hand, we could use the full-size BERT model, which performs better in a lot of benchmark tasks including classification as a bigger model allows better contextual understanding thus better generalization. On the other hand, we could train several models to explore the hyper-parameter space using grid search or random search to find the best combination according to our model.

References

[1] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. *Generalization in NLI: Ways (Not) To Go Beyond Simple Heuristics*. 2021. arXiv: 2110.01518 [cs.CL].

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[3] Nick Ryan Chris McCormick. *BERT Fine-Tuning Tutorial with PyTorch*. 2020. URL: https://mccormickml.com/2019/07/22/BERT-fine-tuning/.

[4] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[5] Chi Sun et al. *How to fine-tune bert for text classification?* Springer, 2019.

[6] Iulia Turc et al. "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models". In: *arXiv preprint arXiv:1908.08962v2* (2019).

[7] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.