Toxic Comment Classification

ELEC-E5550 - Statistical Natural Language Processing D

Henry Kivijärvi 883094, Jade Reinilä 813530, David Sachelarie 101904063

April 2024

1 Introduction

In recent years, natural language processing (NLP) has witnessed significant advancements, particularly in text classification tasks such as toxic comment classification. Identifying and filtering toxic comments in online platforms is crucial for maintaining a healthy and respectful online community. Toxic comment classification has been approached with diverse methods, such as different types of neural networks and transformers.

Singh, Goyal, and Chandel [1] found that using a 3-tier convolutional neural network based on AlexNet architecture yielded promising results with an accuracy of 98.5 %. The study utilized common preprocessing steps for the data, such as tokenization, lemmatization, removing stop words and punctuation, and converting the tokens into embeddings. The proposed CNN model beat all other models in the study in terms of all metrics considered - accuracy, ROC-AUC score, and F1 score. Models under comparison included LSTM and tf-idf based models for example[1]. Another popular approach to toxic comment classification is transformer models. BERT is a language representation model and it stands for *Bidirectional Encoder Representations from Transformers* [2]. Bidirectional architectures can successfully tackle diverse NLP tasks [3], which is why we wanted to see if we would achieve promising results using BERT.

For our course project we chose to participate in the course competition for toxic comment classification and implement our own solution to tackle this task. In this report we outline our approach to toxic comment classification, which entails exploring different models combined with various text preprocessing techniques. Our strategy was to implement three types of models: a baseline model using Support Vector Machines (SVM), a Convolutional Neural Network (CNN) model following AlexNet architecture, and fine-tuned transformer models based on BERT and RoBERTa. We will evaluate and compare the performance of these models and preprocessing techniques, aiming to identify the most effective approach for toxic comment classification. Throughout the report, we will discuss our thought process behind the models and preprocessing choices as well as our findings from evaluating these models.

2 Methods

In the subchapters below, we are diving deeper into the techniques we employed for solving the text toxicity detection task: vectorizing words using **GloVe**, weighing the importance of those words with the **rtf-igm** formula, and classifying the samples using an **SVM**, **AlexNet**, and **BERT- and RoBERTa-based models**.

2.1 Word embeddings: GloVe

Since words are a form of categorical data, they can't be fed directly to a classifier, unless they are converted to a continuous form. Moreover, some words have related meanings or occur in the same contexts, so it would be very useful to represent words in a way which showcases their similarities. A process called *word vectorization* (representing words as abstract multidimensional vectors) is the solution to both aforementioned challenges. Probably the most widely used advanced word vectorization tool is *word2vec*[4], but we opted for the more recent *GloVe*[5], which, according to the paper that introduced it, should achieve better results. GloVe embeddings were used for the baseline and CNN models described in the subchapters below.

2.2 Word weights: rtf-igm

The word vectorization discussed above can further be improved by using the idea that some words in a sample are more "important" than others, so their corresponding vectors should be weighted according to some measure of relevance. This aim is close to what *tf-idf*[6] tries to achieve, i.e. giving importance weights to words in a sample, but adapted to a binary classification task. It is more useful to compare the sample frequency of a word in a class against its frequency in the other class, instead of contrasting its overall sample frequency against the sample frequency of the other words. The formula we used for this aim is *rtf-igm*[7], slightly simplified for binary classification:

$$rtf - igm = \sqrt{tf} \cdot (1 + \lambda \cdot max(\frac{f_0}{f_1}, \frac{f_1}{f_0}))$$

In the formula above, *tf* is the term frequency in the sample, $\lambda = 7$ as suggested in the paper[7], and f_0 and f_1 are the term frequencies in classes 0 and 1, respectively.

2.3 Baseline model: SVM

As a baseline, we used a simple SVM[8], one of the most widely used basic machine learning techniques for binary classification. Its performance gave us a good measure of how important of a contribution can neural networks truly bring in solving this task, and whether their use is really justified.

2.4 CNN model: AlexNet

For our second model, we wanted to implement some kind of neural network. As mentioned in the introduction, we found an article supporting the suitability of convolutional neural networks for toxic comment classification [1]. The article's 3-tier convolutional neural network based on the AlexNet architecture yielded promising results with an accuracy of 98.5 %, so we were inspired to replicate this architecture for ourselves.

Our final model follows the network structure defined in the article excluding the first two layers - the embedding layer and the following dropout layer. We used rtf-igm-weighted GloVe embeddings for the tokens, which we then fed into three adjacent convolutional blocks. The convolutional blocks consist of a convolutional layer, an activation function layer (Exponential Linear Unit), and a max pooling layer. The output of each block is concatenated and fed to a fully connected layer to produce final outputs. The exact structure of the model is described in more detail in the article [1] and is implemented in our code appendix.

2.5 Fine-tuned transformers: BERT- and RoBERTa-based models

In addition to the SVM model and the CNN model, we wanted to assess the performance of existing neural network models from the Hugging Face *transformers* library. The different BERT models seemed to have the most potential for the task at hand. We decided to experiment with both BERT-cased and BERT-uncased, because it seemed reasonable that considering the character casing could be beneficial for our task. We assumed that toxic sentences could have more uppercase letters (e.g. from names of certain personalities).

We decided to rule out BERT-Large due to its computational requirements, even though it is likely the most powerful model among the considered BERT models. However, we decided to include RoBERTa in addition to BERT-cased and BERT-uncased. RoBERTa is also based on BERT, but it utilizes dynamic masking [9], which could be the key for even better results.

Our strategy with BERT-cased, BERT-uncased and RoBERTa was to use the corresponding sequence classifiers and tokenizers on top of the chosen base models. After achieving the final results for each of these models, we decided to include the predictions of the best model in our final ensemble prediction.

3 Experiments

The experiments we performed while implementing the models, as well as the preprocessing techniques we applied, are explained in detail in the subchapters below.

3.1 Preprocessing

The following preprocessing techniques were considered and applied:

- **Data balancing** - Firstly, we noticed that the training dataset was unbalanced, having more datapoints in the non-toxic class, so we decided to implement data balancing. We calculated the number of toxic class datapoints and randomly sampled the corresponding number of datapoints from the non-toxic class. We concatenated those datapoints with the toxic class datapoints and shuffled the whole dataset to make the order of datapoints random. Overall, we achieved better results after implementing the data balancing technique.

- **Tokenization** - Sentences have to be split into words. For this task, we are using *word_tokenize* from *NLTK*[10] and *BertTokenizer* and *Roberta-Tokenizer* from *transformers*[11].

- **Lowercasing** - All uppercase letters are converted to their lowercase counterparts. This is only done for the baseline and CNN models, since this step is already handled automatically by the fine-tuned transformers' tokenizer.

- **Stopword removal** - Stopwords (very frequent words whose meanings are irrelevant on their own) are removed, which ensures that the sample embeddings are not affected by them and provides a good way of reducing the length of sentences before training the transformers.

- **Removal of words with non-alpha characters** - Words which do not contain only letters are likely to be irrelevant, so they are removed in the preprocessing for the baseline and CNN models.

- **Stemming** - Stemming (the truncation of related words to a common stem) is important for ensuring that words with the same meaning but a different form are not regarded as completely different from one another. We thus used stemming for calculating the frequencies of words for the rtf-igm formula. This was the only task we used stemming for though, since the fine-tuned transformers require the full (unstemmed) form of the words they will be trained on, and the original form of the words is also needed for retrieving their GloVe embeddings.

3.2 Baseline model

The baseline model is a basic SVM from *scikit-learn*[12] with default parameters, trained on data preprocessed in two ways. Firstly, after obtaining GloVe embeddings for every word in each sample, we sum those vectors to obtain a single vector for each sample. The resulting training set only contains numbers and is of the dimension (NUMBER_OF_SAMPLES, 300), for a GloVe embedding dimension of 300. The performance of the SVM using the data preprocessed this way wasn't good enough though, so we attempted and managed to improve it by multiplying each word vector with its weight obtained using the *rtf-igm* formula. The word vectors are still added up to a single sample vector, so the resulting training set has the same dimension as above.

3.3 CNN model

The same preprocessing methods as for the SVM were applied to the dataset before training AlexNet as well, with one modification: word embeddings are considered separately for each sample, which after padding and truncation leads to 200 embeddings of size 300 per sample. The resulting dimension is thus (NUMBER_OF_SAMPLES, 200, 300).

To train the CNN model we implemented a training loop following the AlexNet article [1], but also experimented with some different parameters. We found the Adam optimizer used in the article to yield the best results in our implementation as well. We also tried the Stochastic Gradient Descent (SGD) optimizer, but it resulted in worse training results. Thus we stuck with the Adam optimizer in our final implementation. We tried a few different learning rates with Adam and found 0.001 to work the best compared to higher learning rates.

As suggested by the AlexNet article we only used 3 epochs for training the model. Higher numbers of epochs, such as 10 and 20, did not affect the convergence or improve the results. The loss was calculated with *BCEWithLogitsLoss* from *PyTorch*, which combines a sigmoid layer and binary cross entry loss in a single operation as utilizing these operations separately is less numerically stable [13]. Finally, we applied the sigmoid fucntion on the predictions and evaluated the results with F1 score.

3.4 Fine-tuned transformers

During the early stages, we initially had a too high learning rate for the models. The learning rate of 2e-3 resulted in the models learning to predict all sentences as non-toxic. At this stage, we had not implemented the data balancing yet so that could have affected this outcome. When we decreased the learning rate to 5e-5, the performance of the models increased significantly. This also remained true after applying data balancing.

Considering the chosen optimizer, we found AdamW to produce the best results. Initially, we experimented with the default optimizers and Adam, but AdamW beat those during every test run. This is likely due to AdamW using weight decay, which effectively prevents the model from overfitting to the training data.

In addition to the default *transformers* tokenizers, we also used some other preprocessing methods. We first used the *NLTK* library's *word_tokenize* for each sentence. Then we filtered out stopwords from the sentences as an attempt to reduce the length of the sentences. In addition, we considered that stopwords are likely not useful in classifying sentences as toxic or non-toxic.

We also tried some custom preprocessing tricks. We calculated the occurrences of each token in both classes from the training data. Then we picked the most class-specific tokens from each sentence in both the training set and the test set according to the maximum sentence length. We made sure that the original token order remains correct in each modified sentence. We kept this implementation in the final version since we achieved slightly better results with it.

After experimenting with different maximum sentence lengths, we found the length of around 100-125 tokens to be optimal. This decision was based on considering the improvement in the models' performance versus computational costs. We used padding to prevent issues with sentences shorter than the chosen maximum length.

After the preprocessing steps, we fed these modified sentences as input to the models. Finally, we used *scikit-learn*'s softmax function to the models' prediction logits and chose the most probable class for each sentence according to it. Lastly, we calculated the desired metrics including accuracy score and F1 score.

BERT-uncased and BERT-cased seemed to achieve similar performance with all the different experimented data sizes, with BERT-cased being mostly slightly more accurate. RoBERTa achieved the best score of these models when using a small dataset of approximately 20 000 data points. However, it performed significantly worse on the full data compared to BERT-uncased and BERT-cased. We assume that was caused by overfitting despite using AdamW as the optimizer.

Due to computational limitations, we also tried to freeze the base models' layers and only train the layers of the sequence classifiers. However, all the models performed significantly worse with the layers frozen so we were forced to settle for the longer training times. The computational costs and each model's performance could have likely improved by experimenting more with different parameters. We consider the most probable parameters for achieving better results to be learning rate and training data size. However, an exhaustive experimentation on variations in performance using different combinations of parameters was not a viable option for us due to the lack of computational power required to run the models repeatedly.

4 Results

For training our models we used the datasets provided in the Kaggle competition. The training, validation, and test sets had 99,000, 11,000, and 12,000 data points respectively. All data sets have columns 'text' and 'label' to represent the comments that were aiming to classify and their given label of toxic (1) or non-toxic (0). However, in the test data the 'label' column only contains '?', so we can predict those for the competition.

The SVM and AlexNet were trained on a dataset in which, for each sample, unchanged GloVe embeddings were used for each word in a sample (method called "GloVe" in the table below), or additionally by also multiplying each word vector with its corresponding *rtf-igm* weight ("GloVe + rtf-igm" in the table below). Though the second approach had a substantially positive effect on the SVM's performance, it didn't seem to affect much the performance of AlexNet. Both the SVM and AlexNet models are capable of reaching an F1 score of 0.89 and an accuracy of 0.92 on the validation data set. However, their performance is overall worse compared to our transformer models.

We ran all fine-tuned transformer models with the same parameters for a few different training data sizes to be able to compare the results (in table below). Overall BERT-uncased, BERT-cased, and RoBERTa resulted in fairly similar results with only slight differences. When training with the full training data, the BERT-uncased and BERT-cased models outperformed the RoBERTa model both in terms of F1-score and accuracy on validation data. Both of the first two models yielded an F1 score of 0.91 and an accuracy of 0.93. Since RoBERTa is essentially an optimized version of BERT and is trained on more data, we assumed it would perform better compared to BERT-uncased and BERT-cased, which proved not to be the case at least in our case. On the other hand, when training each model with 20,000 data points the BERT-uncased model had the highest accuracy on validation data, 0.93, where as the RoBERTa model had the best results for F1 score, 0.91. It was surprising to us that RoBERTa performed better with a smaller data set than when using the full data. As mentioned in the experiments section, this may be due to some overfitting happening with a larger data set. Overall, all of our fine-tuned transformer models performed quite well and seem to be very suitable for the task of toxic comment classification.

We also tried an ensemble model approach and combined the test predictions of some individual models to see, if this would improve the accuracy of the predictions. We took the most common prediction for each data point and used those as the final predictions. We tried this with RoBERTa + SVM + AlexNet and BERT-uncased + SVM + AlexNet, and in the Kaggle competition both of these combinations received 0.939 test F1 score for the submitted predictions. In the end we got our best test set results with BERT-uncased, BERT-cased, and RoBERTa trained on 20,000 data points. The first two received an F1 score of 0.958 and the third one got a score of 0.956.

Model \ Score	f1 score	accuracy
SVM (GloVe)	0.82	0.87
SVM (GloVe + rtf-igm)	0.89	0.92
AlexNet (Glove)	0.88	0.91
AlexNet (GloVe + rtf-igm)	0.89	0.92
BERT uncased (all data)	0.91	0.93
BERT cased (all data)	0.91	0.93
RoBERTa (all data)	0.81	0.84
BERT uncased (20K)	0.90	0.93
BERT cased (20K)	0.90	0.92
RoBERTa (20K)	0.91	0.88

Table 1: Models' results

5 Conclusions

Since all models we tried performed well, we can conclude that the text toxicity detection task can be tackled successfully, at least for deciding whether a piece of text is toxic or not, without taking into account its degrees or types of toxicity (binary classification task). It also doesn't seem to be a problem for which the use of advanced neural networks is vital, since, even though fine-tuned transformers such as BERT and RoBERTa might have an edge over simpler classifiers, a similar performance can be achieved using a basic SVM with suitable data preprocessing.

Reflecting on our study, more extensive hyperparameter tuning and further exploration of alternative ensemble methods, coupled with accessing additional computational resources, could further refine the models' performance and improve the results.

6 Division of Labor

Work was divided as follows:

- **Henry Kivijärvi** - Implemented the fine-tuned transformer models and their preprocessing, as well as the data balancing technique which was used for all the final models. Researched literature on various promising techniques for modelling the text toxicity detection task.

- **Jade Reinilä** - Implemented part of the basic preprocessing which was used for the baseline and CNN models. Contributed to the implementation of the fine-tuned transformer models and ran the various versions of the aforementioned models. Researched literature on various promising techniques for modelling the text toxicity detection task and found AlexNet, which proved to be a well-performing model for solving this task.

- **David Sachelarie** - Implemented part of the basic preprocessing which was used for the baseline and CNN models, and also implemented the application of rtf-igm to the word embeddings. Implemented the baseline and CNN models. Researched various preprocessing techniques and found rtf-igm, which proved to be a suitable formula for giving importance weights to words in a binary classification task.

References

[1] I. Singh, G. Goyal, and A. Chandel, "Alexnet architecture based convolutional neural network for toxic comments classification," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 2022.

[2] "Transformers documentation - BERT." `https://huggingface.co/docs/transformers/model_doc/bert`. Accessed: 21 March 2024.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *Google AI Language*, 2018.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," 2014.

[6] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, 2002.

[7] K. Chen, Z. Zhang, J. Long, and H. Zhang, "Turning from tf-idf to tf-igm for term weighting in text classification," *Expert Systems With Applications*, vol. 66, 2016.

[8] "sklearn.svm.svc." `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. Accessed: 20 March 2024.

[9] "Roberta." `https://huggingface.co/docs/transformers/model_doc/roberta`. Accessed: 12 April 2024.

[10] "Documentation: Natural language toolkit." `https://www.nltk.org/`. Accessed: 19 April 2024.

[11] "Transformers." `https://huggingface.co/docs/transformers/index`. Accessed: 19 April 2024.

[12] "scikit-learn: Machine learning in python." `https://scikit-learn.org/stable/`. Accessed: 19 April 2024.

[13] "Bcewithlogitsloss." `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`. Accessed: 19 April 2024.

Appendix

The full code can be found here: `https://github.com/dsachelarie/snlp_toxicity_detection`