SI630 Project Report
Toxic Spans Detection

Chenyun Tao
School of Information
University of Michigan
Ann Arbor, MI, 48109
cyuntao@umich.edu

Abstract

Offensive language on social media could make people uncomfortable. To build a healthy online community, human moderators would need to deal with the toxic comments, and NLP techniques like sentiment analysis could help them. Researchers have developed efficient toxic comment classifiers, but do not identify the exact positions of the toxic parts in a text. Here I fine-tune RoBERTa to perform a token classification approach, and propose a toxic spans detection system, which could mark character offsets of toxic spans in the input text. The performance of my model was acceptable (0.661 F1 score), although it still needed improvements. The result shows that token classification could work well to identify the positions of toxic spans in a text. I anticipate experienced researchers could bring more elegant models to detect toxic spans, for example, considering actual toxic spans instead of toxic tokens, and thus people could be more pleasant in online discussions.

1 Introduction

This project aims to solve the Toxic Spans Detection task. Although "several toxicity (abusive language) detection datasets and models have been released", most of them serve as classifier of the whole text, and do not identify which parts of a text are toxic. The Toxic Spans Detection task is to identify the spans (a sequence of words) that make a text toxic. The system solving this task accepts a text as an input, and then outputs a list consisting of the character offsets (starting from 0) of toxic spans in the input text, or an empty list if the text does not contain a toxic span. (John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier, 2021)

I treated this task as mainly a token classification problem, and fine-tuned a RoBERTa to solve it. My model achieved a 0.661 F1 score, which was much better than a random baseline or a offensive list baseline, and was slightly better than a spaCy tagging baseline. The result was acceptable, although the model needed to be improved. Solving this problem could help human moderators to deal with comments, which would reduce their workload, and contribute to a healthy online environment (John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier, 2021). The main contributions of this project are:

- demonstrating that token classification could work well to mark the positions of toxic spans in a text
- proving that fine-tuning RoBERTa with not so many epochs could still result in good results, so it is possible for people with limited computing resources to work on NLP tasks
- showing that learning rate is more important than number of epochs, batch size, and Adam Epsilon in toxic spans detection

2 Data

John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier (2021) provide training, dev (the authors call it trial data) and test data set on Github[1]. They used posts (comments) from the Civil Comments dataset, which contains toxic classification annotations. They kept only the approximately 30k posts that had been regarded as toxic, and randomly picked 10k samples from the 30k posts. Then three crowd-raters from the crowd-annotation platform worked on one post. The annotations might be absent for the cases when marking toxic spans are impossible, such as "the core message being conveyed may be inherently toxic, and,

[1] https://github.com/ipavlopoulos/toxic_spans

hence, it may be difficult to attribute the toxicity of those posts to particular spans".

The data is the same as the one I had for the update, but a bit more description is added here. The training, dev and test dataset are stored as csv files with 2 columns: `spans` and `text`, where the `spans` column consists of the list of the character offsets of toxic spans, and the `text` column consists of the original text. The training data contains 7946 lines of data, the dev data contains 690 lines, and the test data contains 2000 lines. The text in the data sets is not exactly clean, for instance, a few non-toxic spans are labelled as toxic. Fortunately, the authors also provide a `fix_spans` function to fix this problem. In addition, there are no null values in the data, so filling null values is not needed. The last preprocessing step is to convert the data type of spans annotation from string to list, since the original annotations are stored as string in csv files, which is difficult to be utilized. A few examples of the data are shown in Table 1.

Table 1: A few examples of the data

spans	text
[12, 13, 14, 15, 16, 17, 18, 19, 20, 21]	Lol. What a ridiculous insult.
[0, 1, 2, 3]	Damn, a whole family. Sad indeed.
[]	But, but, but, is NOT a defense. It's not even a good deflection. In America today we have Nazis waving the Nazi flag at rallies in our cities. In what capacity does anyone think this is ok and who would not see that as a problem?

It is a bit difficult to see the class distribution of the data, but we could see the distribution of the ratio of toxic characters in the text, which could be calculated as the length of the spans list over the length of the text. The median toxic ratio is 6.8% in the training data, 6.5% in the dev data, and 4.4% in the test data. Distributions of the ratio of toxic characters among all the characters of the text in the test data and the training data are shown in Figure 1, Figure 2 and Figure 3. Scaled versions of histograms including about 95% of the data are shown in Figure 4, Figure 5 and Figure 6. From these figures, it seems that the ratio of toxic characters is higher in the training data set than in the test data set. In addition, most of the text have only a few toxic characters.

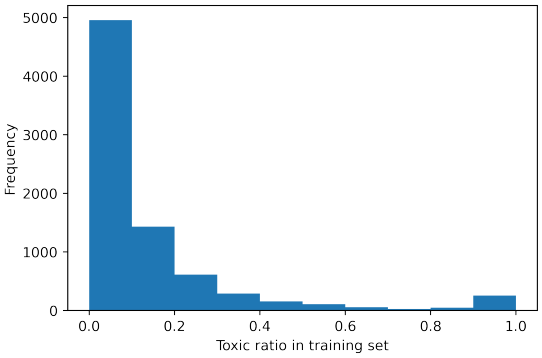

Figure 1: Histogram of ratio of toxic characters in the training data set

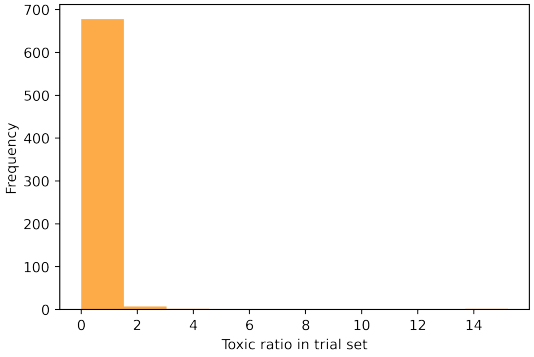

Figure 2: Histogram of ratio of toxic characters in the trial data set

To sum up, some basic statistics of the data are listed in Table 2.

Table 2: Basic statistics of the data

Set	Statistics	Number
training	instances	7946
dev	instances	690
test	instances	2000
training	median toxic ratio	6.8%
dev	median toxic ratio	6.5%
test	median toxic ratio	4.4%

3 Related Work

Just as John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier (2021) have pointed out, researchers have constructed a variety

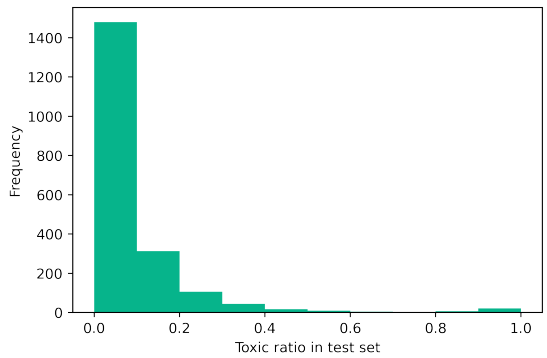

Figure 3: Histogram of ratio of toxic characters in the test data set

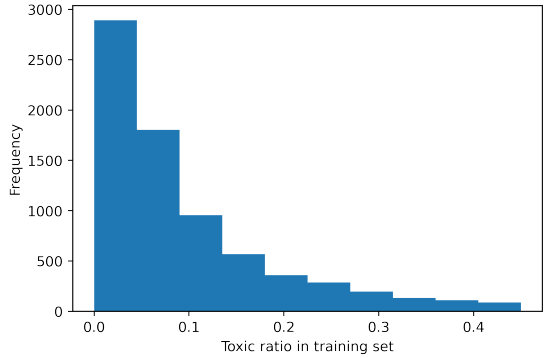

Figure 4: Scaled histogram of ratio of toxic characters in the training data set

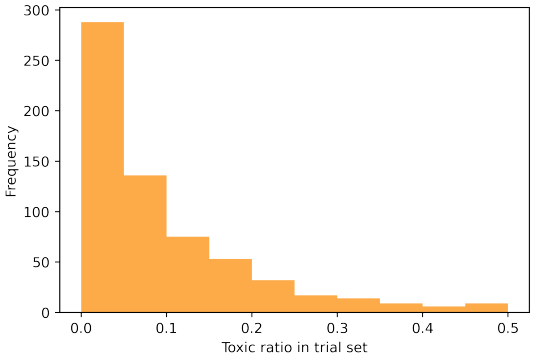

Figure 5: Scaled histogram of ratio of toxic characters in the trial data set

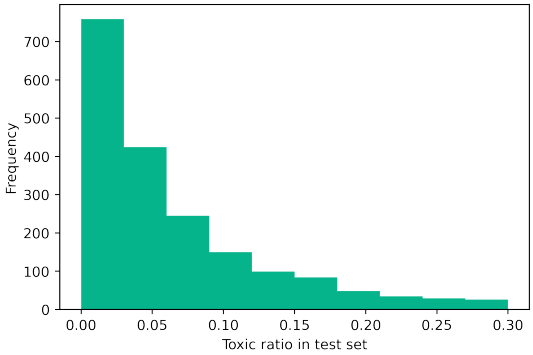

Figure 6: Scaled histogram of ratio of toxic characters in the test data set

of models on toxic comments detection. Pavlopoulos et al. (2017) proposed a classification-specific attention mechanism, and claimed that RNN (Recurrent Neural Network) with their attention mechanism has a good performance. Ziqi Zhang, David Robinson, and Jonathan Tepper (2018) come up with a method combining DNN (Deep Neural Network) with CNN (Convolutional Neural Network) and GRU (Gated Recurrent Units). Mestry et al. (2019) developed classfiers with CNN and fastText word embedding. Anand and Eswari (2019) experimented on LSTM (Long short term memory cell) or CNN with or without GloVe embedding. D'Sa et al. (2020) introduced two methods, and found that within the two methods, "fine-tuning the pretrained BERT (Bidirectional Encoder Representations from Transformers) model" outperformed the approach "extracting of word embeddings and then using a DNN classifier".

These methods proposed by the researchers performed well, and the best performance come from the methods using BERT, but the problems they were solving was different than mine. The difference is that these models mainly deal with classifying the comments, but do not figure out which parts of a text are toxic. While the classification models could be useful for automatic moderation on comments, I think my approach on the Toxic Spans Detection task would be better as semi-automatic moderation is still necessary, and identifying the toxic spans to assist human moderators would work on semi-automatic moderation.

In my project, I fine tune a RoBERTa (Robustly Optimized BERT Pretraining Approach) proposed by Liu et al. (2019), which is an improved retraining of the original BERT, to perform token classification on the text, and mark the positions of the toxic parts.

4 Methods

In this project, I fine tuned a RoBERTa to perform the token classification approach for toxic spans detection, based on `NERModel` in the

simpletransformers[2] library.

4.1 Fine-tuning RoBERTa for Token Classification

From exploratory data analysis, it seems that most of the toxic spans detection problem are just toxic tokens detection. Thus, it seems that toxic spans detection could be simplified as a token-level classification task, where only the offsets of the toxic tokens are marked. As D'Sa et al. (2020) claimed that "fine-tuning the pre-trained BERT model" outperformed the approach "extracting of word embeddings and then using a DNN classifier", and RoBERTa performs better than the original BERT, I think fine-tuning RoBERTa to conduct the token classification task could work well. Figure 7 shows a typical BERT-based token classification model, where a classification layer is added to the output layer.

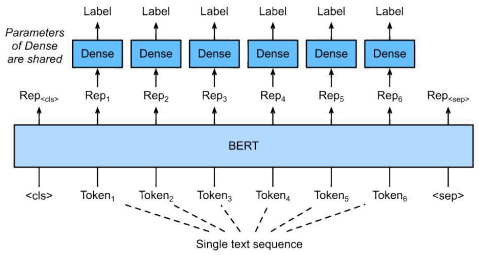

Figure 7: Fine-tuning BERT for token classification[3]

For simplicity, I pick the NERModel from the simpletransformers[2] library to solve the token classification task. I chose the robert-base pretrained model, set batch size to be 4, learning rate to be 3×10^{-7}, number of epochs to be 50, epsilon used in AdamOptimizer to be 10^{-9}, and the other hyperparameters remain default values.

4.2 Data Processing

To feed the data into NERModel, data processing to convert the data into required input formats of theNERModel is still needed after preprocessing in Section 2.

4.2.1 Training and Trial Data Format

One of the acceptable input formats for the training and evaluating data of the NERModel is a Pandas DataFrame containing 3 columns, sentence_id, words, and labels. An example of the required format provided by the simpletransformers documentation is shown in Table 3. To convert the training and dev data from formats in Table 1 to formats in Table 3, I first extract the exact toxic word list from spans. For example, for the first row in Table 1, the corresponding toxic word list should be [ridiculous]. Next, I split the text by space, and remove the leading and trailing punctuation of each word, as punctuation might hinder matching of the words, and does not convey much toxic information. Then nontoxic words are given the label "N", while toxic words in the previous toxic word list are marked as "T". For simplicity, the text in each row is regarded as one "sentence", and thus the sentence_id is just the same as the row id. An example of converting the first row of Table 1 is shown in Table 4.

Table 3: An example of the required input format provided by the simpletransformers documentation

sentence_id	words	labels
0	Harry	B-PER
0	Potter	I-PER
0	was	O
0	a	O
0	student	B-MISC
0	at	O
0	Hogwarts	B-LOC
1	Albus	B-PER
1	Dumbledore	I-PER
1	founded	O
1	the	O
1	Order	B-ORG
1	of	I-ORG
1	the	I-ORG
1	Phoenix	I-ORG
sentence_id	original sentence	
0	Harry Potter was a student at Hogwarts	
1	Albus Dumbledore founded the Order of the Phoenix	

4.2.2 Test Data Format

The required input format of test data is just the text, but we still need to remove the leading and trailing punctuation of each word. To make sure we could extract the exact offsets of words in the text, the original text is also kept. The output format of NERModel's predictions is a list of lists of dicts,

[2] https://simpletransformers.ai/docs/ner-model/

[3] Image source: https://www.d2l.ai/chapter_natural-language-processing-applications/finetuning-bert.html

Table 4: An example of the converted first row of Table 1

sentence_id	words	labels
0	Lol	N
0	What	N
0	a	N
0	ridiculous	T
0	insult	N

where the key of each dict is the word, and the value of each dict is the corresponding tag, for example, [[{'Lol': 'N'}, ...], ...]. I first extract the toxic words, which are the keys corresponding to values of "T". Then I find the offsets of these toxic words in the original text by regular expressions.

5 Evaluation and Results

5.1 Evaluation Metric

John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier (2021) offer the evaluation metric, where F1 score is used. Let system A_i return a set $S^t_{A_i}$ of character offsets of toxic spans in a text t, and G^t be the character offsets of the ground truth annotations of t. F1 score of system A_i with respect to the ground truth G for post t is computed as

$$F_1^t(A_i, G) = \frac{2 \cdot P^t(A_i,G) \cdot R^t(A_i,G)}{P^t(A_i,G) + R^t(A_i,G)}$$

$$P^t(A_i, G) = \frac{\left|S^t_{A_i} \cap S^t_G\right|}{\left|S^t_{A_i}\right|}$$

$$R^t(A_i, G) = \frac{\left|S^t_{A_i} \cap S^t_G\right|}{\left|S^t_G\right|}$$

where $|\cdot|$ denotes set cardinality. If S^t_G is empty for some t, $F_1^t(A_i, G) = 1$ if $S^t_{A_i}$ is also empty, and $F_1^t(A_i, G) = 0$ otherwise. The average of $F_1^t(A_i, G)$ over all the text t of an evaluation dataset T would be a single score for system A_i.

5.2 Baselines and Results

John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier (2021) implement several baselines on Github[1]. I used two of them. One is just a random baseline. I follow the template provided by John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier (2021) to run the random baseline on the test data, which achieves an average F1 score of 0.121. A second one is the `spaCy tagging` baseline, which is a spaCy named entity tagger for toxic spans. The spaCy tagging baseline achieves a quite good average F1 score of 0.659. Based on the suggestions from Prof. Jurgens, I implemented a third baseline, which label all words in the text that are in an offensive word list as toxic spans. I use the full list of bad words and top swear words banned by Google[4] here. This one achieves an average F1 score of 0.236. My model (see Section 4) has an average F1 score of 0.661, which was much better than a random baseline or a offensive list baseline, and was close to a spaCy tagging baseline. The result was acceptable, although the model needed to be improved. The result will be further discussed in section 6.

The box plot of F1 score of the 3 baselines and my model is shown in Figure 8, and average F1 score of the models is shown in Table 5.

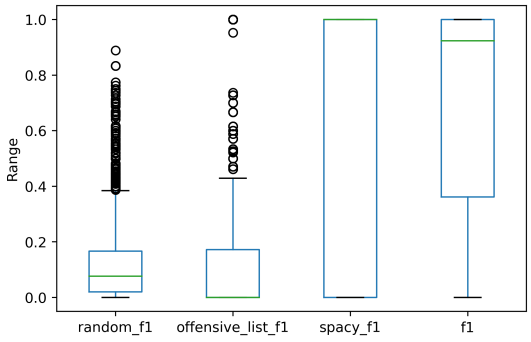

Figure 8: F1-score box plot of the 3 baselines and my model

Table 5: Average F1 score of the 3 baselines and my model

Model	Average F1
random baseline	0.121
offensive list baseline	0.236
spaCy tagging baseline	0.659
my model	**0.661**

5.3 Ablation Study

In this project, I aim to optimize the following 4 hyperparemeters: batch size, learning rate, number of epochs and epsilon used in AdamOptimizer. I tried to use `wandb`[5] to do hyperparameter tuning.

[4] https://github.com/RobertJGabriel/Google-profanity-words

[5] https://wandb.ai/

The hyperparameter sweeping produces several results (shown in Figure 9 and 10), but does not run well, which will be discussed in Section 8 in detail. From these several results, it seems that learning rate is more important than number of epochs and batch sizes.

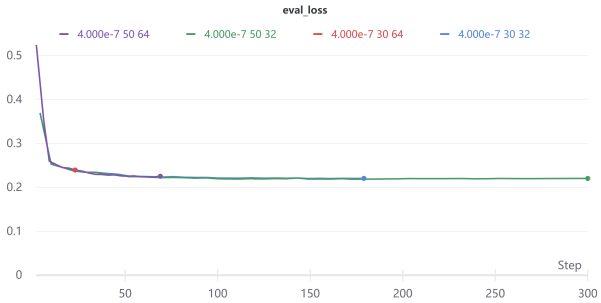

Figure 9: Evaluation loss of models with the same learning rate, and different batch sizes and number of epochs (Legend: learning_rate number_of_epoch batch_size)

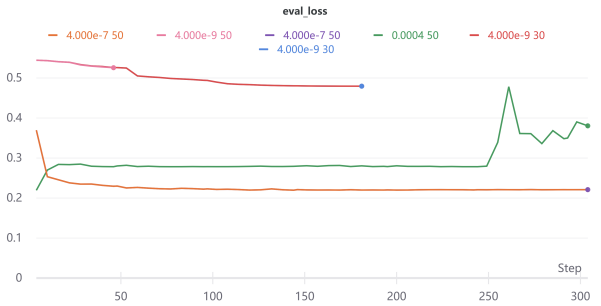

Figure 10: Evaluation loss of models with different learning rates and number of epochs (Legend: learning_rate number_of_epoch)

In addition, I experimented the effect of different combinations of hyperparameters on predictions. The result is shown in Table 6. From this result, it also seems that learning rate is the most important hyperparameter. The best hyperparameter combination here is batch size = 4, learning rate = 3×10^{-7}, number of epochs = 50, epsilon used in AdamOptimizer = 10^{-9}, which has been used as my best model.

6 Discussion

As is stated in the section 5, the random baseline has a low performance on the test data set, with an average F1 score of 0.121, which is not surprising. Random baseline usually could not perform well in text classification tasks, and the data here further decreases the performance of the random baseline. As we have seen in section 2, in most of cases, toxic characters only take up a small proportion of the characters in the text, which indicates that the probability of a character being toxic is actually not high. However, the probability of predicting a character to be toxic is much higher in the random baseline. In addition, the toxic spans consist of consecutive toxic characters, but random baseline would just treat the characters as independent objects, and make random predictions on them. Therefore, we do need models looking into the internal structure of the text.

Table 6: Effect of different combinations of hyperparameters on predictions

Train batch size	Learning rate	Epochs	Adam Epsilon	Mean F1
4	**3e-7**	**50**	**1e-9**	**0.661**
8	3e-7	50	1e-9	0.660
8	4e-7	50	1e-9	0.659
8	5e-7	50	1e-9	0.659
32	4e-7	50	1e-9	0.656
8	4e-7	50	1e-6	0.659
8	3e-7	30	1e-9	0.653
8	3e-6	30	1e-9	0.639
8	3e-4	30	1e-9	0.162
8	3e-7	50	1e-8	0.658

The offensive list baseline also has a low performance, with an average F1 score of 0.236, but is better than the random baseline, which is expected. Unlike the random baseline, this baseline takes consecutive toxic characters into consideration, leading to a higher F1 score. However, the small offensive list size does not support good predictions on the test data, as the list that I used only contain 451 words. Lists with more offensive words might improve the performance, but there exist far more words in the comments to be analyzed, and gathering these offensive words might be labor consuming. Thereby, we do need models learning from the data instead of enumerating offensive words.

The spaCy tagging baseline has an unanticipated high performance on the test data set, with an average F1 score of 0.659. This is a huge improvement to the previous baselines, and is sort of unexpected as using `spaCy` library with few customized steps could result in good results. However, as spaCy is a strong library supporting deep learning workflows for production usage, and keeps updating to the start-of-art, it is reasonable for this baseline to have

a relatively high performance.

Finally my own model also has a relatively high performance on the test data set, with an average F1 score of 0.661. The performance is only slightly better than the spaCy tagging baseline, while I expected the difference between the performance of spaCy tagging baseline and my model to be larger. This is partly because the spaCy tagging baseline works much better than expected. More importantly, as I only try to optimize 4 hyperparameters, and the data size is not large when fine-tuning the model, it is reasonable that my model will not have an extremely high performance, and thus we may need to try to optimize more hyperparameters. However, my model still has a relatively high performance, and is much better than the first two baselines as expected. In addition, from Figure 8, we can see my model has the highest lower bound of F1 score, which means it could produce reasonable results in general. Hence, the performance should be satisfactory for an end-user of my NLP model. Choosing appropriate hyperparameters like learning rate and batch size do help in building a good model.

7 Conclusion

In this project, I fine-tune RoBERTa to perform a token classification approach for the toxic spans detection task, and develop a model that could mark character offsets of toxic spans in the input text. The source code is uploaded to Github at `https://github.com/tcy1999/SI630`. My model has an average F1 score of 0.661, which was much better than a random baseline or a offensive list baseline, and was slightly better than a spaCy tagging baseline. The result is acceptable, although the model still needs improvements, where considering actual toxic spans instead of toxic tokens, tuning more hyperparameters and using other pretrained models might help (more details about future directions will be included in Section 9).

8 Other Things I Tried

During the process of building a working model, I found the most time consuming part was not completing the skeleton of the model, but was trying new approaches and debugging. Sometimes strange errors would happen, for example, the spacy tagging baseline provided by the authors did not run first due to a recent upgrade of spaCy, and finally some of the attempts succeeded, but unfortunately some failed.

As is described in Section 5, I tried to use `wandb` to do hyperparameter tuning, which produces several results, but does not run well. It took me a while to learn to use `wandb` sweeping, and running the sweeps was time consuming, so I also learnt to submit slurm jobs through sbatch on Great Lakes. However, unfortunately, the sweep usually suddenly crushed after 7-8 runs, and every run after it would just failed. I also spent time on debugging it, but it turns out that the failed sweeps were probably due to limited computing resources, as the output log never showed errors.

Another thing I tried was to use spaCy tokenization instead of spliting and removing leading and trailing punctuation. I also spent some time learning the basic usage of spaCy, and it took me a while to overwrite the previous data converting functions. However, the performance of my model decreased a lot, indicating this approach was unsuccessful, and I had to recover the data converting functions.

9 What I Would Have Done Differently or Next

Due to time constraints and limited computing resources, I did not try other pretrained models. I believe adopting more state-of-art and more task-specific pretrained models like SpanBERT proposed by Joshi et al. (2020) instead of just using `roberta-base` would be a meaningful attempt.

Another idea that I would like to try is to consider actual toxic spans or phrases instead of just words. When doing exploratory data analysis, I found while token classification took the major part in toxic spans detection, toxic spans lists actually annotated positions of toxic phrases like “violent and aggressive”, so the toxic spans could not be totally regarded as individual toxic tokens. Therefore, it should be beneficial to consider toxic phrases instead of just solving the simplified token classification task.

In addition, from the results, it also seems that further tuning the hyperparameters might improve the performance. From Table 6, we can see that different combinations of hyperparameters could lead to different average F1 score values. Besides the 4 hyperparameters that I have tested, there still exist quite a few hyperparameters. We may even try to set different learning rates for different layers in the model.

References

Mukul Anand and R. Eswari. 2019. Classification of abusive comments in social media using deep learning. In *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, pages 974–977.

A. G. D'Sa, I. Illina, and D. Fohr. 2020. Bert and fasttext embeddings for automatic detection of toxic speech. In *2020 International Multi-Conference on: "Organization of Knowledge and Advanced Technologies" (OCTA)*, pages 1–5.

John Pavlopoulos, Ion Androutsopoulos, Jeffrey Sorensen and Léo Laugier. 2021. Toxic spans detection.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Suresh Mestry, Hargun Singh, Roshan Chauhan, Vishal Bisht, and Kaushik Tiwari. 2019. Automation in social networking comments with the help of robust fasttext and cnn. In *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pages 1–4.

John Pavlopoulos, Prodromos Malakasiotis, and Ion Androutsopoulos. 2017. Deeper attention to abusive user content moderation. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1125–1135.

Ziqi Zhang, David Robinson, and Jonathan Tepper. 2018. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *The Semantic Web*, pages 745–760, Cham. Springer International Publishing.