Toxic Spans Detection

Sriram Ananthakrishna[*]

[*] Applied Mathematics (Data Science), Stevens Institute of Technology

I. INTRODUCTION

To encourage constructive online debates, moderation is essential. Even though several toxicity (abusive language) detection datasets and models have been made public, the majority of them categorize entire comments or papers rather than identifying the specific passages that make a text hazardous. However, highlighting such toxic spans can help human moderators who frequently deal with lengthy comments and who prefer attribution over just a system-generated unexplained toxicity score per post (such as news portal moderators). Thus, an essential first step toward effective semi-automated moderation is the evaluation of systems that could pinpoint toxic spans within a text. [1]

Previous research on hate speech detection has focused on classifying entire sentences as toxic or not. However, this approach does not provide any information about why a sentence was classified as toxic, which makes it difficult for human moderators to understand the data. A more informative approach is to identify the specific spans of text that are responsible for the toxicity of a sentence. This is known as "toxic spans detection." The Toxic Spans Detection dataset, which is part of the SemEval-2021 Task 5, is a useful dataset that can be used to train models for toxic spans detection.

Toxic spans detection is a challenging task, but it is an important one for promoting healthy online discussions. By identifying the specific spans of text that are toxic, we can better understand the nature of hate speech and take steps to prevent it.

I used the Toxic Spans Detection dataset to train different machine learning models for detecting harmful language postings. My model used **Electra** as the tokenizer and the embedding layer, built 2 models; BiGRU and BiLSTM on top of CRF layer to detect the toxic spans in the input. On the test set, the model achieved an F1 score of 46%.

The rest of this paper is organized as follows. Section II discusses the problem formulation. Section III presents the methodology of Data extraction and cleaning. Section IV discusses the experiments and modeling. Finally, Section V discusses the conclusion and further road map ahead to carry this work forward.

Index Terms- Toxic Spans, Named Entity Recognition, Tokenization, Embedding, Transformers, BiGRU, CFR, TFElectraModel

II. PROBLEM FORMULATION

A literature review was done to understand the problem statement in a better way. The dataset is provided from the SemEval-2021 Task 5: Toxic Spans Detection (Pavlopoulos et al., 2021) [5]. It includes the training and the test sets. Both consist of two parts: the content of texts and the spans denoting the toxic words in the posts. Spans represent toxic words in the posts as a set of character indexes.

The train data contains the column ‘text’ which contains multiple spans of toxic words. For each span, it contains a single word, a phrase, or a sentence.

The input to toxic spans detection is a text, and the output is a list of spans that are toxic. A span is a sequence of tokens. For example, the input "This is a toxic comment" might have the output [(0, 4), (7, 11)], which means that the toxic spans are "This is a" and "comment".

The task is to predict these character offsets given the text. The dataset is provided with English texts with gold annotations of toxic spans and it could be addressed as supervised sequence labeling. The training must be done on the provided posts with gold toxic spans. It could also be treated as rationale extraction, using classifiers trained on larger external datasets of posts manually annotated as toxic or not, without toxic span annotations.

Named entity recognition (NER), code-switching detection, quotation detection, and key-phrase extraction are among many tasks that involve span identification.

The objective of the project is to create a classification model which classifies each word whether it is toxic or not on a high level. In depth, the model uses NER (Named-Entity Recognition) and applies a LSTM to achieve this task.

III. METHODOLOGY

A. Data Preprocessing

The following Data preprocessing is done on the data:

1. We firstly transform spans into a set of words.
2. Then, we pre-process the posts as by segmenting the posts by the ElectraTokenizer from transformers developed by Huggingface.
3. Changing texts to lower case
4. The function ‘create_inputs’ is used to prepare a batch of text data. The model expects the input data to be in a specific format, and the ‘create_inputs’ function ensures that the data is in the correct format.
5. ‘create_output’ function is executed so that the input to the model comes from this function.
6. We use the ‘Electra’ word embedding to construct the dictionary and encode the text of posts. Posts are encoded by the dictionary of the word embedding.
7. To make sure all vectors are the same length, we add the pad token. Then, we set the maximum length of vectors equal to 400. Spans are transformed into a one-hot vector corresponding to each word in posts where toxic words are denoted as 1 and others are denoted as 0.
8. A set of helper functions which helps the model to apply the NER on the tokenized words.
9. Finally, a snippet of code to split the data into train and validation dataset.

B. Model Design

BiGRU-CRF is a deep neural model used for Named-entity recognition task.

We implement this model for the task of detecting toxic words in documents. The model includes three main layers:

1. The word representation layer uses embedding matrix from the Electra word embedding
2. The BiGRU layer for sequence labeling
3. The Conditional Random Field (CRF) layer to control the probability of output labels.

The output is a binary vector, in which each value determines whether a word is toxic or non-toxic.

The model architecture is shown in the below image.

C. CRF Layer and Loss Function

Conditional random fields (CRFs) are a type of statistical model that are used for sequence labeling tasks, such as named entity recognition (NER).

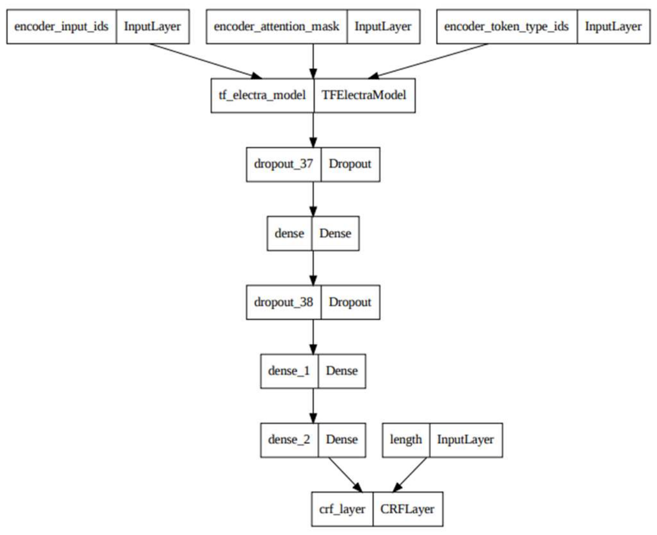
In NER, the goal is to identify the named entities in a text. Named entities are words or phrases that refer to real-world entities, such as people, places, organizations, and events. CRFs can model the dependencies between the labels in an NER task by considering the previous and next labels in the sequence. For example, if the current label is "Person," the CRF can consider the previous and next labels to determine if the current label is likely to be correct.

Here are some of the advantages of using CRFs for NER:

1. CRFs can model the dependencies between the labels in a sequence.
2. CRFs can achieve state-of-the-art results on a variety of NER datasets.
3. CRFs are relatively easy to train and to use.

Here are some of the disadvantages of using CRFs for NER:

1. CRFs can be computationally expensive to train.
2. CRFs are sensitive to the choice of features.
3. CRFs are difficult to interpret.



encoder_input_ids | InputLayer
encoder_attention_mask | InputLayer
encoder_token_type_ids | InputLayer
tf_electra_model | TFElectraModel
dropout_37 | Dropout
dense | Dense
dropout_38 | Dropout
dense_1 | Dense
dense_2 | Dense
length | InputLayer
crf_layer | CRFLayer

Fig 1: Model Architecture of BiGRU-CRF

For the loss function, I am using the in-built loss function used in the CRF wrapper function. To train a CRF, we need a loss that optimizes the model so that the gold-label sequence (the ground truth) has a probability that tends to 1. That loss is obtained by computing the negative log-likelihood of the gold sequence.

$$\begin{aligned} loss &= nll(\mathbf{y}, \mathbf{x}) \\ &= -log\left(\frac{\exp\left(\Sigma_{t=1}^{n} U(y|x) + \Sigma_{t=1}^{n} T(y_t|y_{t-1})\right)}{Z(\mathbf{x})}\right) \end{aligned}$$

Where the numerator is the likelihood of the ground truth sequence. Since we have all emission and transition scores, it is trivial to calculate this likelihood given the labels.[2]

However, computing the denominator (the partition function) is a lot more challenging since there is an exponential number of possible sequences (in the number of tokens in the sequence).

D. *Training and Inference*

The complete architecture of the model is as follows:

1. Word representation layer from the embedding matrix from Electra as the base model
2. Bi-Directional GRU with 512 nodes
3. A dropout layer of 0.1
4. A Dense layer with relu activation function
5. A dropout layer of 0.1
6. A dense layer with relu activation function
7. A final dense layer with linear activation function

The F1-score over the dataset is defined as:

$$\frac{1}{|T|}\sum_{t}^{T} F_1^t = 2 * \frac{P^t(A,G) * R^t(A,G)}{P^t(A,G) + R^t(A,G)}$$

In the above equation, Pt determines the precision, and Rt determines the recall of the post t. The precision and recall are calculated as:

$$P^t(A,G) = \frac{|S_A^t \cap S_G^t|}{S_A^t}$$

$$R^t(A,G) = \frac{|S_A^t \cap S_G^t|}{S_G^t}$$

The St in both the above equations is set of toxic characters of post t (span**).**

IV. EXPERIMENTS

A. *Hardware Requirements*

The training and the evaluation of systems were performed on Google Colab's free GPU. The training time varies with the models. For each model, it is around 40-45 min for training which is well within the 12-hour limit of Colab.

B. *Choosing Models and Hyperparameter*

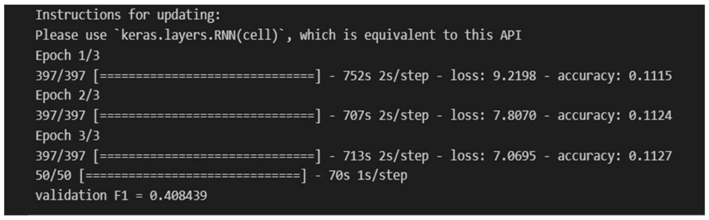
For CRF, I used the Keras-CRF (Hironsan, 2017) layer. I used a batch size of 16, train for 3 epochs, used a linear learning rate, and an Adam optimizer. The initial learning rate is 3e−5. During tokenization, the maximum length allowed is 400.

C. *Results*

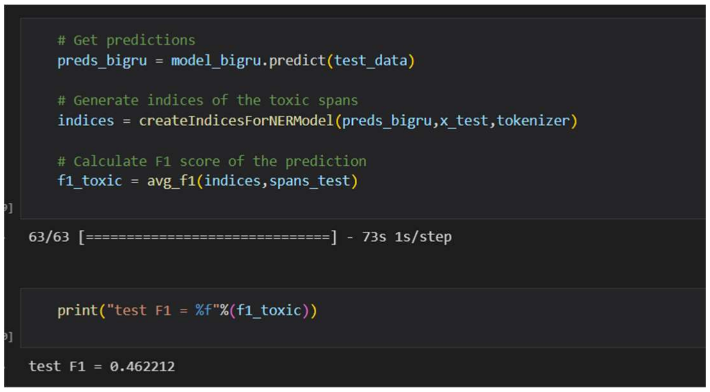
I tried GRU, BiGRU, LSTM and BiLSTM models on Hugging-Face's Electra transformer. Out of these, BiGRU model gave the best training f-1 score of above 0.408. I also tried changing the transformer to Roberta, BERT. By comparison, Electra was performing better than the other two. The other models were constantly in the f-1 range of 0.35 to 0.40 or worse.

Similar was the case when I experimented with different tokenizers. I tried with Electra, BERT, and Auto Tokenizers. Better results were given with Electra tokenizer + TFElectraModel (transformer) embedded + BiGRU-CRF model and Electra tokenizer + TFElectraModel (transformer) embedded + BiLSTM-CRF model

Embedding	**Layers**	**F1(validation)**	**F1(testing)**
Electra	BiGRU-CRF	0.4084	0.4622
Electra	BiLSTM-CRF	0.3947	0.4595



```
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
Epoch 1/3
397/397 [==============================] - 752s 2s/step - loss: 9.2198 - accuracy: 0.1115
Epoch 2/3
397/397 [==============================] - 707s 2s/step - loss: 7.8070 - accuracy: 0.1124
Epoch 3/3
397/397 [==============================] - 713s 2s/step - loss: 7.0695 - accuracy: 0.1127
50/50 [==============================] - 70s 1s/step
validation F1 = 0.408439
```



```
    # Get predictions
    preds_bigru = model_bigru.predict(test_data)

    # Generate indices of the toxic spans
    indices = createIndicesForNERModel(preds_bigru,x_test,tokenizer)

    # Calculate F1 score of the prediction
    f1_toxic = avg_f1(indices,spans_test)

63/63 [==============================] - 73s 1s/step

    print("test F1 = %f"%(f1_toxic))

test F1 = 0.462212
```

The Validation F-1 score was about 0.4084 and Testing F-1 score was 0.4622.

V. CONCLUSION AND FUTURE WORK

I used the BiGRU-CRF model and its variants for detecting toxic words in the sentences. My model achieves 40.8% by F1-score from the analysis, it is found that GRU based model performs better than LSTM based models. The reason behind the assertion might be an area of interest. Also, Using Bi-directionality in the models only minutely improved the performance of the models. All in all, my model predicts just average and there is more room for improvement. By further researches, we can surely improve the performance of the detection model by applying the attention mechanism and using the character-level representation combining with word-level representation. Character-level models like CharBERT (Ma et al., 2020) is a potential approach to increase the performance of toxic spans detection tasks.

REFERENCES

[1] Dataset https://competitions.codalab.org/competitions/25623.

[2] https://hyperscience.com/blog/exploring-conditional-random-fields-for-nlp-applications/

[3] https://www.davidsbatista.net/blog/2017/11/13/Conditional_Random_Fields/.

[4] Son T. Luu , Ngan Nguyen , UIT-ISE-NLP at SemEval-2021 Task 5: Toxic Spans Detection with BiLSTM-CRF and ToxicBERT Comment Classification https://aclanthology.org/2021.semeval-1.113.pdf

[5] John Pavlopoulos , Jeffrey Sorensen, Leo Laugier, Ion Androutsopoulos, SemEval-2021 Task 5: Toxic Spans Detection.

[6] Gunjan Chhablani, Abheesht Sharma, Harshit Pandey, Yash Bhartia, Shan Suthaharan, NLRG at SemEval-2021 Task 5: Toxic Spans Detection Leveraging BERT-based Token Classification and Span Prediction Techniques.

[7] Alireza Salemi, Nazanin Sabri, Emad Kebriaei, Behnam Bahrak, Azadeh Shakery, UTNLP at SemEval-2021 Task 5: A Comparative Analysis of Toxic Span Detection using Attention-based, Named Entity Recognition, and Ensemble Models.