BennettNLP at SemEval-2021 Task 5: Toxic Spans Detection using Stacked Embedding Powered Toxic Entity Recognizer

Harsh Kataria, Ambuje Gupta, Vipul Mishra*
Department of Computer Science Engineering
Bennett University, Greater Noida, India
(hkataria99, ambujegupta99)@gmail.com
vipul.mishra@bennett.edu.in

Abstract

With the rapid growth in technology, social media activity has seen a boom across all age groups. It is humanly impossible to check all the tweets, comments and status manually whether they follow proper community guidelines. A lot of toxicity is regularly posted on these social media platforms. This research aims to find toxic words in a sentence so that a healthy social community is built across the globe and the users receive censored content with specific warnings and facts. To solve this challenging problem, authors have combined concepts of Linked List for pre-processing and then used the idea of stacked embeddings like BERT Embeddings, Flair Embeddings and Word2Vec on the flairNLP framework to get the desired results. F1 metric was used to evaluate the model. The authors were able to produce a 0.74 F1 score on their test set.

1 Introduction

Modernization has awarded us with the technology capable of communicating with masses across huge distances in an instant by the touch of a single finger in our palms, the smartphone. With all this innovation every day, it is now more accessible than ever, even in the most rural parts of the world and at very reasonable costs. It has enabled a vast population to share their thoughts and views on popular topics in public forums. People can express themselves in the most creative ways and talk to each other about movies, research, politics, the economy and much more. Some of the key forums and platforms for such activities are Facebook, Twitter, YouTube, Reddit. They allow users to participate in various discussions, discussions that at times may not be very decent. This creates an issue for these platforms as they usually have some of the other policies against indecent content posted by users. On average there are about 350,000 tweets, 510,000 comments, 293,000 status updates on Facebook and Twitter in every 60 seconds (Sayce, 2020). It is humanly impossible for these platforms to check each and everything posted by the users for hate or toxicity. They require an automated method to flag such content.

The motivation for this research task is to achieve some degree of automatic moderation in the social web. It is crucial to moderate social media sites such as Facebook, Twitter and Reddit to be healthy and inclusive. This includes filtering and censoring toxic and hateful content posted online on these public forums. There are automated hate detection NLP models like (Zhang and Luo, 2018) capable of identifying toxic content with acceptable performance. However, they do not identify the specific spans of text that are toxic. This task tries to identify these spans that can be used further to provide insights into a generic text toxicity score. More about the study that this paper is based on can be found from the task organizers paper (Pavlopoulos et al., 2021).

This paper proposes linked lists for data preprocessing and a stacked embedding approach to training this automated system.

The authors' experiment and code for the models ready to be reproduced can be found using the authors' Github repository.

This paper is organized as follows. It starts with a short abstract describing the paper at a very high level. Then the first section introduces the problem at hand and the task to accomplish. It mentions the authors' approach and the link to their code and models. The second section talks about the background research performed for the task and explains the existing solutions and how this paper is different from them. The third section gives an in-depth understanding of the system approach to solving the tasks where it talks about the embeddings and the stacking method. The fourth section tells us about the dataset, how it was presented, what it consisted of, the data processing applied to it, the experimental setup and the evaluation metrics. The fifth section is about the methodology, followed by the results section. The seventh section concludes the system description. Finally, we get the references.

2 Background research and related works

This problem tackles the challenge of accurately identifying the parts of a toxic text and contributing to making the complete text toxic. There, however, exist systems that can score the toxicity of a full text, sentence or comment like (Zhang and Luo, 2018), but they don't identify the exact part of that sentence that is toxic. There also are systems that can accurately say if a statement is toxic or not and even go on to the extent of identifying the emotion projected by it, for example, if it is positive, negative, humorous, sarcastic, offensive or funny and even the level of these emotions as seen in (Gupta et al., 2020).

A widely known methodology to identify parts of a text is Named Entity Recognition (NER) (Yamada et al., 2020) and its types like the speech (POS) tagging method. NER is a sequence labelling task that recognises entities as per the types of entities it is trained on. These are usually nouns or particular words like names of people or places or organisations and values like monetary numbers and currencies. The sequence labelling task considers a text sequence in which part of it needs to be tagged differently according to the embeddings. Long short term memory, LSTMs (Hochreiter and Schmidhuber, 1997) are usually used for sequence labelling. A variant of LSTMs, the bi-directional recurrent neural network-based BiLSTM, has proved to be very successful at performing accurately at such tasks. It is also seen to be combined with the conditional random field(CRF) decoding layer as seen in (Ma and Hovy, 2016) to get better results.

Open-source tool, Spacy (Honnibal et al., 2020) is a leading tool to create effective and lightweight NER models for datasets with such challenges.

3 System Overview

For this task, the authors have used the flairNLP platform (Akbik et al., 2018a) to stack BERT and flair embeddings and create a Named Entity Recognition (NER) model. They have morphed the original aim of the NER type tasks to train a model capable of identifying toxicity in a text that is already classified as toxic as a whole. This was possible because, finally, we were using the different embeddings related to toxic parts of the text. These were the embeddings that were trained, which made the task very similar to a NER task. The model created also had a conditional random field layer. Its purpose was to better understand the context around the text's selected parts by considering the neighbouring words.

For this task of sequence labelling, we have stacked embedding by concatenating BERT and flair embedding. This helps in getting a better semantic context out of the concatenated embedding word vectors. Following is a brief about flair and BERT embeddings.

3.1 Flair Embedding

Flair embedding (Akbik et al., 2018b) are a type of contextual embedding in which sentences are passed as character into a character level language model to generate word embedding. Contextual string embedding are generated through LSTM because of its ability to keep long term dependency within their hidden state. In Figure 1

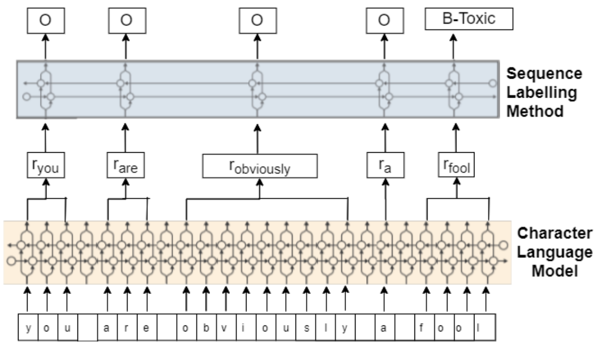

Figure 1: A word is passed as characters to generate flair embedding. After that a CRF layer is used to convert this into a NER problem.

We can see each character of the text ”you are obviously a fool” is passed through a bidirectional character-level neural language model, and each word is retrieved. Then it is given to the CRF layer(sequence labelling). Following is a brief explanation of the mathematics behind generating these embedding. In LSTM, the hidden state h_t represents the past sequence of the character. Both forward and backward language model is used to create these embedding. The mathematical equation of the hidden layers of the forward and back-

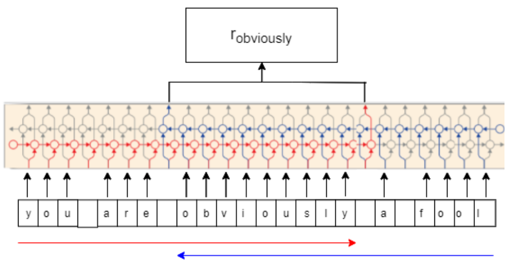

Figure 2: Generating Contextual String Embeddings for the word ”obviously”. In the forward language model(represented in red), the output hidden state is extracted after the last character in that word. For the backward Language Model(represented in blue), the output hidden state is extracted before the word’s first letter. These two output layers are concatenated to form the final embedding

ward model can be seen in equation 1 and 2 respectively.

$$h_t^f = f_h^f(x_{t-1}, h_{t-1}^f, c_{t-1}^f, \theta) \tag{1}$$

$$h_t^b = f_h^b(x_{t+1}, h_{t+1}^b, c_{t+1}^b, \theta) \tag{2}$$

In the above equation, f and b are the forward and backward model’s notations. Memory cell and parameters of the model are represented by c_t and θ, respectively. The output of the hidden state from both the forward and backward language model are concatenated. In the forward language model, the hidden state output is extracted after the word’s last character. Subsequently, for the backward language model, the hidden state’s output is extracted before the first character of the word. Both the language model captures the semantic information and then are concatenated to generate the word embedding for that particular word. Let us suppose individual word string begin with $t_1...t_n$ then the contextual word embedding can be seen in equation 3

$$w_i^{charLM} = \left\{ \begin{array}{c} h_{t_{i+1}-1}^f \\ h_{t_i-1}^b \end{array} \right\} \tag{3}$$

To better understand the concept, figure 2 explains the process by taking an example, and the forward language model is shown in red colour. The backward language model is shown in blue colour. To have a complete understanding of how the contextual word embedding work, one can refer to the (Akbik et al., 2018b) paper.

3.2 BERT Embedding

Unlike Flair embedding, BERT embedding (Devlin et al., 2019) are word-level embedding, and it only contains the encoder part of the transformers (Alammar, 2018). In this paper, the authors have used 2 BERT models, i.e. BERT-base, which contains 12 layers, produces an output of 768 units and BERT-Large, which contains 24 layers, produces an output of 1024 units. Generating word embedding can also be classified as feature extraction in which embedding are generated and are fed into the neural network. For the BERT-base model, each layer produces 768 units for every word. To generate BERT embedding, authors have concatenated the last 4 hidden layers. BERT paper (Devlin et al., 2019) also shows that concatenating the last 4 layers yield the best results. BERT has around 30,000 words vocabulary. If the word is not in the vocabulary, then the BERT tokenizer converts it into a sub-word or characters. For example - Word “unrecognized” will be represented as [’un’, ’re’, ’co’, ’gni’, ’zed’]. The first token of a sentence is always ([CLS]), which indicates the sentence’s starting. Two separate sentences are separated with ([SEP)] tokens. An example of how the BERT model works is shown in figure 3 which is adopted from (Alammar, 2019) where authors have used ”you are obviously a fool” as an example sentence. We can see first the sentence is converted into the sentence convention used by BERT, i.e. adding the required tokens. After adding the tokens, BERT tokenization is used, i.e. words are converted into numbers by referring to the BERT dictionary. BERT tokenization also converts any words in sub-words or character if the required word is not present in the vocabulary file. The ids are then fed into the BERT model, and desired output, i.e. BERT embedding, are generated. Each layer produces 768 units for a single word. The last four (4) hidden layers have been concatenated to get the word embedding in the paper. Now we can put these words embedding to our model to generate results. Figure 4 shows

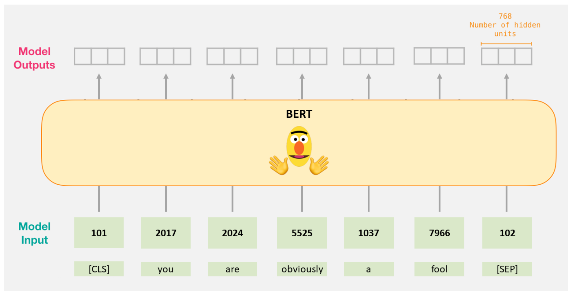

Figure 3: Example Illustrating the process of generating BERT Embedding.

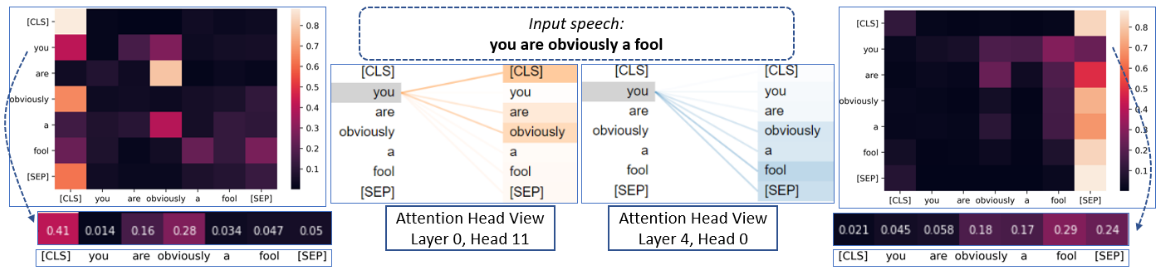

Figure 4: Visualization of Attention in BERT-Base model. Lighter points in the heatmap represents more attention value

the visualization of attention using BERT. The heat map shows how BERT correlates one word with other.

4 Experimental Setup

This section presents a brief overview of the data, the pre-processing it went through and challenges faced in doing so, information about the experiment environment and the evaluation metrics used.

4.1 Data

The data was provided to the authors by the task organizers who had acquired it from the Civil Comments Dataset (Borkan et al., 2019). The task organizers then filtered all the text level toxic-labelled text that had been labelled toxic by more than half the annotators, which was around 30,000 in number from a total of 1.2 million posts. Out of these 30,000 text posts, random 10,000 posts were selected and given to crowd annotators to mark the toxic spans from the text. More information about this can be found in the task description paper (Pavlopoulos et al., 2021). The authors finally received a CSV file with two columns with headers as 'spans' and 'text'. Spans column contained a list of character level indices of the toxic entities. Next to it was the complete text that was found to be toxic. For some texts, there existed a corresponding list of empty spans if no toxic span was annotated. Authors randomly collected 558 data points from the dataset (CSV) as a testing set and were left with 5339 data points as the training set.

4.2 Linked-list based Pre-processing

The pre-processing stage involved coming up with a method to map the spans before and after cleaning the text. The data provided had a column full of rows with toxic text collected from social media and hence it was naturally in need of cleaning as it contained a lot of abbreviations, punctuation, some foreign characters, numbers and special characters that were not supposed to be present there as they would create ambiguity to the model. Removing these would bring some uniformity to the model input. In this approach authors faced majorly two (2) challenges.

1. Removing unwanted characters from the original text will produce a cleaner text and that text will be shorter in length as compared to the original text. This will create discrepancy from the spans column, as the spans are given was according to the length of the original sentence.

2. While pre-processing it was important to maintain the sequence of the words in a sentence. For example "You are an Idiot", we have to make sure that after the first word "You" the second word is "are" only.

To tackle the first problem we replaced unwanted(punctuation, numerals etc.) characters with whitespace character so that the length of the sentence remains the same.

Text : You.... are, idiot!!

Pre-processed text : You   are idiot

Ground spans = [12,13,14,15,16] = ['idiot']

In the above text, we can see that the unwanted characters are replaced by whitespace character which solves our first problem.

For the second problem, we implemented a linked list data structure. After tokenizing the sentence on whitespace we stored individual word in a single node. The head of the node is attached to the next node which helps us to maintain the sequence

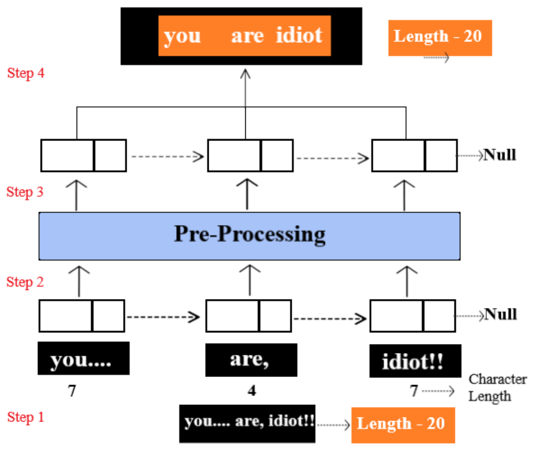

Figure 5: Implementation of linkedlist data structure for pre-processing

of words in a sentence, hence solving our second problem.

This is described stepwise in figure 5

Step 1: The original sentence is tokenized on ”whitespace”. In the example, the length of the original text is 20.

Step 2: After tokenization, each word is stored in a node of a linked list data structure and each word from each node is pre-processed i.e. unwanted characters are removed.

Step 3: To maintain the sequence of words another linked list is made but in this linked list each word is cleaned. The unwanted characters are replaced with whitespaces in the pre-processing block.

Step 4: Words are joined back to form a proper sentence, and the length of the new sentence is 20 as it was of the original sentence.

After getting the pre-processed text, it is ready to be fed into the model.

4.3 Experiment Environment

The experimental environment set up by the authors included the use of Python, mostly for scripting along with some of the well known and commonly used python libraries like NumPy, pandas, flair(flairNLP), re(regex). They used Jupiter notebook for python ide along with python version 3.7. Google Colab (GPU and TPU) was also used for training the models.

4.4 Evaluation metrics

Authors have used F1 score (Da San Martino et al., 2019), precision and recall (Goutte and Gaussier, 2005) in order to evaluate the performance of the models. The task organisers also used F1 score to evaluate and rank the challenge responses. Equation 4 represents the mathematical formula of calculating the F1 score.

$$F_1{}^l = \frac{2 \cdot P^l(X_i, Y) \cdot R^l(X_i, Y)}{P^l(X_i, Y) + R^l(X_i, Y)} \tag{4}$$

In equation 4, F_1^l represents F1 score of the system i, which is calculated for the text l. The predicted values are represented by X_i whereas the ground truth is represented by Y. P and R represents Precision and Recall values with their mathematical calculations shown in equations 5 and 6 respectively where S represents the Set function.

$$P^l(X_i, Y) = \frac{|S^t_{X_i} \cap S^t_Y|}{|S^t_{X_i}|} \tag{5}$$

$$R^l(X_i, Y) = \frac{|S^t_{X_i} \cap S^t_Y|}{|S^t_Y|} \tag{6}$$

Here, $|\cdot|$ represents Cardinality which can be interpreted as the length of the finite set.

The overall F1, precision and recall of the models' performance was obtained by calculating the mean of individual scores of every text in our test set of 558 data points.

5 Methodology

In this paper authors have used the concept of stacked embeddings i.e. to generate a particular embedding for a word, flair gives you the flexibility to concatenate different word embeddings together to get better results. Equation 7 depicts concatenating flair and GloVe embedding.

$$we_i = \left\{ \begin{matrix} we_i^{Flair} \\ we_i^{GloVe} \end{matrix} \right\} \tag{7}$$

Here word embedding is denoted by we. In this paper, the result of four (4) models have been displayed. Table 1 shows different parameters used by authors in different models. The concept of early stopping and adaptive learning rate are used to generate these results. The learning rate would be halved if the model does not show improvement consecutively 4 times in a row. In this case, the training automatically stops when the learning rate becomes too small for example LR=6.2500e-05.

Parameters	Model 1	Model 2	Model 3	Model 4
Epochs	30	39	30	36
Learning Rate	0.001	0.001	0.001	0.001
Mini Batch Size	8	8	8	8
Embeddings	Flair,BERT-large-uncased, CharacterEmbeddings	Flair,BERT-large-uncased, CharacterEmbeddings,GloVe	Flair,BERT-base-uncased, CharacterEmbeddings,GloVe	Flair,BERT-base-uncased, CharacterEmbeddings

Table 1: Parameters of different Proposed Models

6 Result

The parameter details of the 4 different models and the embeddings used are listed in table 1. It lists the number of epochs the model was trained on with the initial learning rates and batch sizes. Model 2 ran for the most epochs before being auto-stopped as no improvements were seen in the last 4 epochs and the learning rate parameter got too small due to the adaptive learning rate function. All the models had Flair and Character embeddings with the variation of GloVe and BERT-uncased embeddings. Table 2 lists different models used

Model	F1	Precision	Recall
Model 1	0.748	0.971	0.929
Model 2	0.737	0.967	0.925
Model 3	0.726	0.968	0.916
Model 4	0.724	0.973	0.909

Table 2: Experimental results

and their respective F1-scores, precision and recall values for our test set. It can be inferred that model 1 i.e. stacked embeddings with Flair, BERT-large-uncased, CharacterEmbedding performed the best with an F1 score of 0.748 with precision and recall of 0.971 and 0.929 respectively. It was able to predict toxic spans closest to the ground truth. The other models are not too behind than this but it seems that model 4 did not have too much of a difference when stacked with GloVe embeddings as seen in model 3.

Figure 6 presents the comparative analysis of F1 scores achieved by the 4 models proposed by the authors in table 1 and the best performing model from the NLRG system (Chhablani et al., 2021) and the UniParma system (Karimi et al., 2021). The authors of NLRG have used a BERT based RoBERTa token classification method to reach their best F1 score of 0.689. The authors of UniParma have used CharacterBERT and the bag of words(BOW) method to get their F1 score of 0.66. Proposed model 1 (highlighted in red) was the best performing model with a 0.748 F1 score.

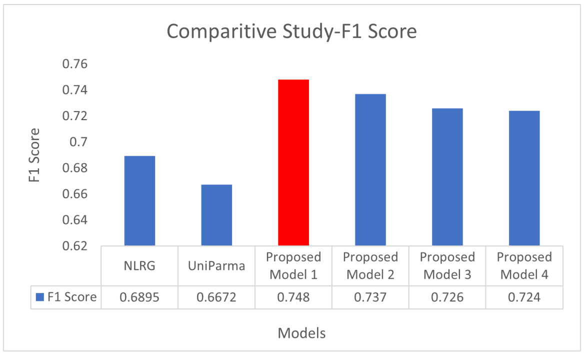

Figure 6: Comparative Analysis of F1 scores

Text : You.... are, idiot !!

Pre-processed text : You are idiot

Ground spans=[12,13,14,15,16]=['idiot']

Predicted spans=[12,13,14,15,16]=['idiot']

The above example displays the input text, the pre-processed clean text with the toxic word ”idiot” highlighted in red. Below them is the ground truth for the spans of this toxic word and then there are the correct predicted spans for it.

7 Conclusion

We support the systematic development for identifying toxic spans. We have successfully been able to deploy a linked list approach to prepare the data and then train it using the stacked embeddings method and produce empirical results. This task can prove to be useful in providing a better analysis in the censoring of toxic posts on the internet.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018a. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018b. Contextual string embeddings for sequence labeling.

Jay Alammar. 2018. The illustrated transformer.

Jay Alammar. 2019. A visual guide to using BERT for the first time.

Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. 2019. Nuanced metrics for measuring unintended bias with real data for text classification. *CoRR*, abs/1903.04561.

Gunjan Chhablani, Yash Bhartia, Abheesht Sharma, Harshit Pandey, and Shan Suthaharan. 2021. Nlrg at semeval-2021 task 5: Toxic spans detection leveraging BERT-based token classification and span prediction techniques.

Giovanni Da San Martino, Seunghak Yu, Alberto Barrón-Cedeno, Rostislav Petrov, and Preslav Nakov. 2019. Fine-grained analysis of propaganda in news article. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5640–5650.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding.

Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *Proceedings of the 27th European Conference on Advances in Information Retrieval Research*, ECIR'05, page 345–359, Berlin, Heidelberg. Springer-Verlag.

Ambuje Gupta, Harsh Kataria, Souvik Mishra, Tapas Badal, and Vipul Mishra. 2020. Bennettnlp at semeval-2020 task 8: Multimodal sentiment classification using hybrid hierarchical classifier. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation, SemEval@COLING 2020, Barcelona (online), December 12-13, 2020*, pages 1085–1093. International Committee for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Akbar Karimi, Leonardo Rossi, and Andrea Prati. 2021. Uniparma @ semeval 2021 task 5: Toxic spans detection using characterBERT and bag-of-words model.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. pages 1064–1074.

John Pavlopoulos, Léo Laugier, Jeffrey Sorensen, and Ion Androutsopoulos. 2021. Semeval-2021 task 5: Toxic spans detection (to appear). In *Proceedings of the 15th International Workshop on Semantic Evaluation*.

David Sayce. 2020. The number of tweets per day in 2020.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. Luke: deep contextualized entity representations with entity-aware self-attention. *arXiv preprint arXiv:2010.01057*.

Ziqi Zhang and Lei Luo. 2018. Hate speech detection: A solved problem? the challenging case of long tail on twitter. *Semantic Web*, Accepted.