Natural Language Processing Midterm Report: Automated Hate Speech Detection

Xiaoyan HU
x.hu@mpp.hertie-school.org

Abstract

Due to the harmful social impact of hate speech content, leading social network companies have designed policies to remove those content. Currently, one conventional way of keeping the platform "clean" is to hire human moderators according to a feature from online media. While the contractors are working under enormous pressure and invest many efforts, their efficiency cannot compare to the explosive increase of hateful content on social media. Furthermore, the moderators must cope with the blatant exposure to traumatic images, out-cry racism, violent xenophobia, and other aggressive content, which introduces extreme mental pressure upon the persons. Thus, this manual labour is not sustainable and should be replaced by more efficient and less costly technological approach.

Recent years, researchers dive into an alternative to manual moderation: automated hate speech detection. The Natural Language Processing (NLP) and Machine Learning (ML) communities worked together closely and came up with classical methods for text classification such as Naive Bayes, Logistic Regression and Random Forest. The improvement of optimization techniques made it possible to explore neural networks and develop deep learning methods on the basis of that. Some of the most popular approaches include Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), typically Long Short-Term Memory network (LSTM) (Shanita Biere, 2018)[3].

This team project aims at understanding the mechanism behind automated hate speech detection by building a deep-learning detector with the dataset from the open web. Thus, it is duplication work fundamentally. We apply state-of-industry RNN approach for classification task on a Twitter dataset. Now, this project has managed to preprocess the dataset and generated the features with Tfidf model. We are still working on debugging the RNN model and will try to apply the classical models in the final report for comparison.

1. Proposed Method[1]

1.1. Defining hate speech

The diversity of hate speech definition challenges the interoperability of hate speech detection method because those differences decide which aspect of the speech to identify. Since our project is not going to collect and label our own dataset, we can only accept the hate speech definition of the selected dataset. We decided to proceed with our exploration with the dataset constructed by Davidson et al. (2017)[4]. They defined hate speech as the language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group. It is important to note that Davidson's hate speech data does not include some highly contextual and frequently used phrase such as "b*tch" or f*g even though they are prevalent on social media. It is a pity that those words are excluded as they constructed the everyday life insults that many people suffer online.

1.2. Deep Learning Model: Recurrent Neural Network (RNN)

A common verdict of the RNN approach is that it is good at modelling units in sequence. RNN is often compared with CNN as the latter is supposed to be good at extracting position invariant features (Yin, 2017)[5]. Wenpeng Yin, Katharina Kann, Mo Yu and Hinrich Schutze compared CNN with popular RNN models such as Gated Recurrent Unit (GRU) and Long Short-Time Memory (LSTM). In their brief introduction of the three models, Yin et al. described the LSTM as a word sequence model with three gates: input gate it, forget gate ft and output gate ot. All gates are generated by a sigmoid function over the ensemble of input xt and the preceding hidden state ht1. In order to generate the hidden state at current step t, it first generates a temporary result qt by a tanhnon-linearity over the ensemble of input xt and the preceding hidden state ht1, then combines this temporary result qt with history pt1by input

[1]GitHub page link:https://github.com/Xiaoyan-Adele/Automated-Hate-Speech-Detection

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{x}_t\mathbf{U}^i + \mathbf{h}_{t-1}\mathbf{W}^i + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{x}_t\mathbf{U}^f + \mathbf{h}_{t-1}\mathbf{W}^f + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{x}_t\mathbf{U}^o + \mathbf{h}_{t-1}\mathbf{W}^o + \mathbf{b}_o) \\
\mathbf{q}_t &= \tanh(\mathbf{x}_t\mathbf{U}^q + \mathbf{h}_{t-1}\mathbf{W}^q + \mathbf{b}_q) \\
\mathbf{p}_t &= \mathbf{f}_t * \mathbf{p}_{t-1} + \mathbf{i}_t * \mathbf{q}_t \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(\mathbf{p}_t)
\end{aligned}
$$

Figure 1. mechanism of LSTM

gate it and forget gate ft respectively to get an updated history pt, finally uses output gate ot over this up-dated history pt to get the final hidden state ht (2017).

2. Experiments

2.1. Data and Data Pre-processing

Before applying any machine learning models, our project first inspects the data and making sure that they are feasible for analysis. Out cloned dataset has a sample of 24,802 labelled tweets stored in CSV format. After importing the CSV into Google Colab, we relied mostly on the NLTK library for text preprocessing. We replaced the URLs, excessive white spaces and mentions with a single gap. Then we tokenized the text and remove punctuations and set to lowercase. After that, this project stemmed the tokenized text and pasted the preprocessed text back to the labelled dataset.

2.2. Feature Generation and Training Testing Split

Transforming a natural language into n-grams is vital for calculating the counts of words in sentences and subsequently change the text into numerical data. One of the most common methods to achieve this transformation is TF-IDF. In Natural Language Processing with Pytorch, it separates the TF-IDF into two sub-concepts:

- TF representation of a phrase, sentence, or document is simply the sum of the one-hot representations of its constituent words.
- Inverse-Document-Frequency (IDF) representation penalizes common tokens and rewards rare tokens in the vector representation.

In our experiment, we used the sklearn library to apply TfidfVectorizer to our processed dataset and them used the train_test_split function to split the complete dataset into two categories: training (70%) and testing (30%). We noticed that the Tfidf vectorizer only works on text. In order to get the score, we had to put tokens back to a string, which is intuitively contradictory to the common practice in the text preprocessing part. Thus, we will inspect this contradiction further to understand the reasons behind.

$$
w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)
$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figure 2. TFIDF function

3. Future work

At this point, we are still in the process of constructing and testing the LSTM model with our dataset. Thus we cannot report any precision, recall or F1-score to as to demonstrate our progress. As for work in the future, we still continue debugging the RNN model and will try to apply classical model in the final report for the purpose of comparison.

References

[1] MacAvaney S, Yao H-R, Yang. E, Russell. K, Goharian. N, Frieder. O (2019) *Hate Speech Detection: Challenges and Solutions*. PLoSONE 14(8):e0221152. https://doi.org/10.1371/journal.pone.0221152

[2] Pinkesh Badjatiya1, Shashank Gupta, Manish Gupta and Vasudeva Varma (2017). *Deep Learning for Hate Speech Detection in Tweets*. International World Wide Web Conference Committee (IW3C2),published under Creative Commons CC BY 4.0 License.WWW ACM 978-1-4503-4914-7/17/04. http://dx.doi.org/10.1145/3041021.3054223

[3] Shanita Biere (2018). *Hate Speech Detection Using NaturalLanguage Processing Techniques*. Research Paper, Vrije Universiteit Amsterdam

[4] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. (2017). *Automated Hate Speech Detection and the Problem of Offensive Language*. Proceedings of the Eleventh International AAAI Conference on Web and Social Media

[5] Wenpeng Yin, Katharina Kann, Mo Yu and Hinrich Schutze (2017) *Comparative Study of CNN and RNN for Natural Language Processing*. arXiv:1702.01923v1 [cs.CL] 7 Feb 2017