Team Peter-Parker at SemEval-2019 Task 4: BERT-Based Method in Hyperpartisan News Detection

Zhiyuan Ning, Yuanzhen Lin, Ruichao Zhong
Department of Computer Science and Technology
Beijing Normal University - Hong Kong Baptist University United International College
Zhuhai, P.R.China
{chineseperson5, jeremy0077jj, answer980810}@gmail.com

Abstract

This paper describes the team peter-parker's participation in Hyperpartisan News Detection task (SemEval-2019 Task 4), which requires to classify whether a given news article is bias or not. We decided to use JAVA to do the article parser and the BERT model to do the bias analysis and prediction. Furthermore, we will show experiment results with analysis.

1 Introduction

As the Hyperpartisan News Detection is getting more and more popular in NLP area in recent years, our team decided to focus on such kind of topic and choose task 4 in 2019 SemEval competition, which requires to decide whether a given news article is showing an unreasoning or blind allegiance to some specific groups or persons(Kiesel et al., 2019). For the task, it also requires the competitor's model to classify the news article in one of the two classes, bias or not. In previous SemEval competition, the classification tasks were mostly regarded as sentiment analysis on Twitter, news or scientific paper and so on. For SemEval 2019, the task focus on bias detection, which gives great help for people in daily life to acquire the news and articles in a more objective way.

So far, the machine learning approaches to do the bias detection were mostly using the RNN(Iyyer et al., 2014) or the Word Vectors(Anil Patankar and Bose, 2017), which can get the accuracy for more than 70%.

To reach a greater performance, we decided to adopt a state-of-the-art language model, BERT (Devlin et al., 2018), which set new records on many NLP tasks recently, into our political bias task analysis.

For dealing with the given large dataset, JAVA was used as a parser tool to help us make those training articles more readable.

2 Model Description

Our task is to predict whether an article or news is bias or not, which is entirely a binary classification task. Recently, Google released an essay about BERT and its code. They also provided the performance of BERT model on different tasks in the essay. One of them is similar to our task, which is called SST-2. It is a binary classification task, which is The Stanford Sentiment Treebank, a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment. According to the result (about 95% accuracy), BERT perform pretty well on SST-2 task.

We chose to use BERT to do the task since it had better result on binary classification. We will introduce the BERT model and its detailed implementation in this section. In the following part, we will introduce the model architecture, input representation, pre-training procedure, and fine-tuning procedure.

2.1 Model Architecture

The architecture of BERT is a multi-layer bidirectional Transformer encoder. In this model, it indicates the number of layers as L, the hidden size as H, and the number of self-attention heads as A. It set the feed-forward/filter size to be 4H. It also provides two model sizes.

- BERT-Base: L=12, H=768, A=12, Total Parameters=110M
- BERT-Large: L=24, H=1024, A=16, Total Parameters=340M

2.2 Input Representation

No matter the input in one token sequence is a single text sentence or a pair of text sentences, BERT input representation is capable of representing them. To construct the input representation of a given token, we merged the corresponding token, segment and position embeddings.

2.3 Google pre-trained BERT

Instead of using traditional left-to-right or right-to-left language models, Google pre-train BERT using two new unsupervised prediction tasks which are Masked LM and Next Sentence Prediction.

2.4 Fine-tuning Procedure

It is easy to do BERT fine-tuning for sequence-level classification tasks. By construction corresponds to the special [CLS] word embedding, we take the final hidden state (i.e., the output of the Transformer) for the first token in the input for the interest of obtaining a fixed-dimensional pooled representation of the input sequence. We denote this vector as C $\in R^H$. Additionally, the only new parameters added during fine-tuning are for a classification layer W $\in R^{K*H}$. (K is the number of classifier labels). The label probabilities P$\in R^K$ are computed using a standard softmax, P = softmax(CW^T). All the parameters of BERT and W are fine-tuned cooperatively to maximize the log-probability of the correct label. Some modification must be done slightly on the above procedure in a particular task manner for span-level and token-level prediction tasks.

3 Experiments

3.1 Parse on the Datasets

A good training data cannot be made without data cleaning. There were 600,000 training data (by publisher) and 150,000 validation data (by publisher) and 645 training data (by article). For such a huge dataset, we first split data into 75 separated files, each contained 10,000 news articles so that they were easy to be opened by text editors.

After doing this, we used JAVA to do data parsing and cleaning and BERT model to do bias analysis and prediction. In the articles, some of the characters are escaped as HTML such as that " & " became &amp. It was really easy to unescape them with Java, which involved only one line of method invocation: StringEscapeUtils.unescapeHtml4().

There were some unknown Unicode characters in the articles, so it is good to be removed. Unfortunately, when applying some regular expressions to the articles to remove those characters, some of the articles in other languages would be gone, for example, Chinese, since it was hard to find all the occurring unknown characters in all news articles and we had to use a simple method which was blindly removing all the words, not in the range of \x00 to \x7F in Unicode. So, all the unknown characters could not be removed in order for retaining meaningful words in other languages.

Another problem was that most of the articles contained too many urls and a number of html tags because these articles are parsed from the internet. These might affect the performance and accuracy, so we used a set of regular expressions to catch and remove all the urls and html tags in different forms. Finally, another regular expression was applied to remove the duplicate punctuation such as a line of only periods to divide the articles.

3.2 Models and Training

After cleaning the data, we used BERT to train it with two procedures, which are pre-training and fine-tuning procedures.

3.2.1 Pre-training Procedure

According to Google, BERT needs plenty of time and resources to finish the pre-training procedure. Google used 4 Cloud TPUs in Pod configuration (16 TPU chips total) to train BERT-Base model. Considering the limited time and resource, we decided to download the pre-training model from Google instead of training it by ourselves. For the reason that the data consisted of different languages and Google only released the base model for multilingual data, we could only choose this, which is shown below:

- BERT-Base, Multilingual Cased: 104languages, L=12, H=768, A=12, Total Parameters=110M

3.2.2 Fine-tuning procedure

3.2.2.1 Modify processor

BERT needs an explicit input to train or predict, and it contains the processor to process the input of the model. For our task, we created a new processor for the dataset.

For a model that needs to perform a complete process of training, cross-validation, and testing, our custom processor needed to inherits the DataProcessor, overloads the get_labels function to gain label and also overloads the get_train_examples function, get_dev_examples function and get_test_examples function to get the individual input. These are invoked in the main function flags.do_train, flags.do_eval, and flags.do_predict phases, respectively. The contents of the three functions are much the same, except that you specify the address of the file to be read into.

Modifying get_train_examples function, the function needed to return a list which is made up by InputExample class. The InputExample class only contained the initialization functions. The initialization function only needed the variable guid, which was used to distinguish each example and it could be defined as train-%d'%(i) way. text_a is one string, text_b is another string. Text_a and text_b are two strings and they were merged with [CLS] and [SEP] to become [CLS]text_a[SEP]text_b[SEP]. This merged string was then given to the model. The last parameter, label, was also a string, and it should be guaranteed to appear in the list returned by the get_labels function.

Functions get_dev_examples and get_test_examples were modified using the same method above.

3.3 Results

The prediction procedure was done with the TIRA(Potthast et al., 2019) machine provided by the organizer.

Prediction on Training Set

We split the training set to get the validation set and the test set. The first experiment contained 10000 training articles, 2000 validation articles, 2000 test articles. Also, the proportion of the five categories was equal in the three sets mention above. All the articles are received from them in order from the given training set. Figure 1 shows the result of our first experiments.

Prediction on Test Set

In the second experiment, we directly used the aforementioned model to predict the given test set (byarticle) since the result was so good in the first experiment. But the accuracy was dropped down to 0.6077. The results is shown on the figure 2.

Prediction on Final Test Set

The accuracy we obtained in the second experiment was so bad and we thought that it was due to a lack of training data. So, we anticipated that more training data would help. Next, we trained all the data and predicted the test set pan19-hyperpartisan-news-detectionby-article-test-dataset-2018-12-07. The result was even worse in this final experiment, which is shown in the figure 3. The accuracy dropped to the lowest point, 0.5031. It was just like random guessing.

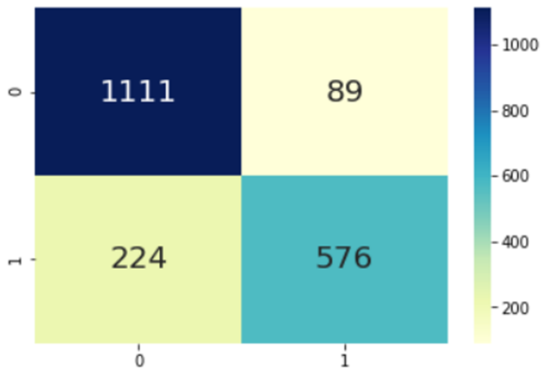


Figure 1: First experiment accuracy:0.9125, 0 is not bias,1 is bias.

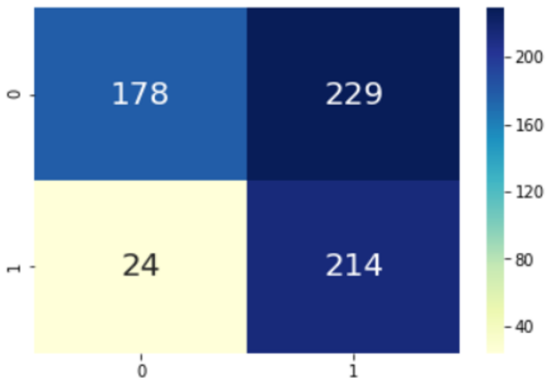


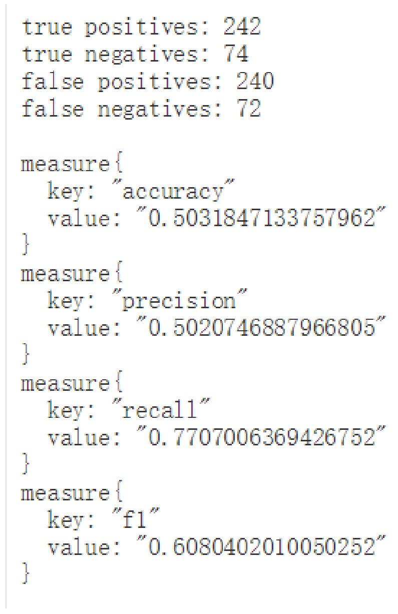
Figure 2: Second experiment accuracy:0.6077, 0 is not bias,1 is bias.

3.4 Error Analysis

We guess the reason why we get two different accuracies in two kinds of articles (by publisher, by article), is that the source of these two articles are different. Data by publisher is decided by the press and publisher, and the other is decided by manual selection. We only trained the data by publisher, so, it seems that it will not perform well on the type of data by article.

4 Conclusion

It was a great experience to use BERT to do the hyperpartisan news detection task although the result was not quite promising compared with the



```
true positives: 242
true negatives: 74
false positives: 240
false negatives: 72

measure{
  key: "accuracy"
  value: "0.5031847133757962"
}
measure{
  key: "precision"
  value: "0.5020746887966805"
}
measure{
  key: "recall"
  value: "0.7707006369426752"
}
measure{
  key: "f1"
  value: "0.6080402010050252"
}
```

Figure 3: Final experiment accuracy:0.5031.

one that Google has achieved. There are many works we can improve in this task, for example, we may do text summarization before training the data since every article is very long. All in all, more effort still need to be spent in the future time.

References

Anish Anil Patankar and Joy Bose. 2017. Bias discovery in news articles using word vectors. pages 785–788.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Mohit Iyyer, Peter Enns, Jordan L. Boyd-Graber, and Philip Resnik. 2014. Political ideology detection using recursive neural networks. In *ACL*.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. SemEval-2019 Task 4: Hyperpartisan News Detection. In *Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval 2019)*. Association for Computational Linguistics.

Martin Potthast, Tim Gollub, Matti Wiegmann, and Benno Stein. 2019. TIRA Integrated Research Architecture. In Nicola Ferro and Carol Peters, editors, *Information Retrieval Evaluation in a Changing World - Lessons Learned from 20 Years of CLEF*. Springer.