Automatic Recognition of Machine Generated text

1800017821 李云济

November 15, 2020

Abstract

In this paper, we report the results of a Machine Generated text recognition task. We utilize a pre-trained Bert model as the base, and fine-tune it for our specific task. Experiments on dev set show that our model can complete the task decently and achieve competitive or better performances than some previous methods.

Key Words: Automatic Recognition, Machine-Generated text, Bert

1. Introduction

It is generally agreed among researchers that the importance of information, or data, can never be stressed enough in our learning procedure. However, the fact is that, there is a lot of fake data around us, which is generated by machine, especially when today's advanced deep learning technology has allowed machine to generate extremely similar sentences with human speech. Therefore, the recognition of machine-generated text becomes a subject worth studying. The high accuracy discrimination of machine generated sentences can also instruct machine to produce more vivid text. This may sound simple but in reality, identifying machine-generated text is a non-trivial problem that has drawn a large body of research in the natural language processing community.

2. Overview

Recently, neural models have been widely used for natural language processing tasks, for their ability to minimize the effort in feature engineering and their outstanding result, and pre-trained language models have also achieved great successes in various NLP tasks, due to their capacity to capture the deep contextualized information in text by pre-training on large-scale corpora. In addition, among all the various neural models, Bert is one of the most effective and impressive ones. Bert and his improved versions have occupied the top of all kind NLP rankings, so in our task, we will also introduce a pre-trained Bert model and fine-tune it with our data.

Our dataset contains a train and two tests, total about 100,000 labelled sentences in the train file, 10,000 labelled in test1, which is also used as the dev set, and 10,000 unlabelled in test2. The length of one sentence can range from 200 to 5000. All the data files contain exactly half written sentences and half machine generated sentences. The human written sentences are labelled as 1, and those generated by machine are labelled as 0. The labels are tagged at the end of each line, so in Python we can simply use line[-1] to extract the label and line[:-1] to extract the text.

3. Our method

1. Pre-trained BERT

Since the sentences are presented as string of words, yet the input of our model has to be numbers, we import the BertTokenizer firstly to transform the lines to vectors. It will also help us build an attention_mask vector for each sentence, which will indicate whether a number is a padding or is derived from a real word. The tokenizer is already pre-trained on its vocabulary. All the code related to neural models is written with PyTorch. The package we mainly use is Transformers, created by Hugging Face.

Then we import BertForSequenceClassification, a Bert model for sequence classification task, because our task is also a kind of sequence classification. To get the logits of classification result, it adds a Dropout layer and a Linear layer after the output of basic Bert. The basic structure of Bert is vital, but we choose not to interpret it here in every detail. Generally speaking, Transformer is a phenomenal and epochal language model, which profoundly harnesses the potency of attentions, especially self-attentions, and Bert is a Bidirectional Encoder Representations from Transformers, with Masked Language Model and Next Sentence Prediction as training tasks. It is the perfect pre-trained model for multiple NLP tasks by now. Our pre-trained model type is bert-base-uncased, which means L=12, H=768, and A=12. The maximum sequence length that this model might ever be used with is 512, which means some sentences will exceed the max length. However, we believe that 512 is enough for our model to tell whether a sentence is machine generated or not.

2. Fine-tune

In the fine-tuning step, we fix the first several layers of our pre-trained model and only adjust the weights of its last four layers, which are ‘layer.10’, ‘layer.11’, ‘bert.pooler’, and ‘out.’. This is because we reckon that the front layers have already captured the contextualized information they need to know, and the fewer weights we have to change, the faster our training will be.

We train our model on one NVIDIA GTX 1080Ti GPU with 11GB memory. We also use PyTorch Dataset and DataLoader to help manage our data. The batch size is 16, because of the shortage of memory. Data is shuffled during each epoch. We train it for 16 epochs, considering our lack of computing ability.

The model’s output of a line is a vector of two numbers, and the index of the higher number represents our answer label. The loss function is CrossEntropyLoss, which is an appropriate loss function for a classification task. It combines logSoftmax and NLLLoss together. At first, we employ the Adam optimizer with default learning rate and betas. However, our loss doesn’t drop even after 5 epochs, so we switch to SGD with default learning rate, which brings some notable results. The training lasts for about 30 hours.

4. Results

Our model gets 0.8365 accuracy on the dev set, and 0.7714 precision, 0.9564 recall and 0.8540 F-

score of human written sentences respectively, and in test set, we didn't find any apparent error. We can notice that the recall is much higher than the precision, which indicates that the model is prone to acknowledge it is written by human, when meeting a lifelike sentence. By reading the dev text manually, we find that some sentences with label 0 also have clear logic and rich meaning, while others may seem obviously generated by machine. In general, we can say that the scores of our model are quite satisfactory, and we hope it can gain a proper result on test set, too.

5. Conclusion

This report presents a suitable model for machine generated text recognition, which adopts Bert as the foundation and fine-tune it for our downstream task. The result on dev set is terrific and beyond our expectation. Nevertheless, the work is just an adaptation of existing methods, and much work needs to be done to evaluate this approach more thoroughly.