HKU-COMP3361-Assignment1

Name: Zhiheng LYU

Email: 3035772432

UID: 3036772432

1. Motivation

Previous research have shown that training from scratch on a large enough dataset can just as well as pre-training [1]. So I tried two different models: one is Bert from scratch. The other uses N-gram to extract features and uses the random forest that is most commonly used on Kaggle for machine learning.

2.Dataset analysis

The training data set is not balanced, and there are some differences in the distribution under different labels.

In train set, there are `140015` words in `['at', 'in', 'of', 'for', 'on']` in total， and `[('at', 9070), ('in', 44981), ('of', 57030), ('for', 13794), (on, 15140)]`, respectively.

In vaild set, there are `14634` words in `['at', 'in', 'of', 'for', 'on']` in total， and `[('at', 1005), ('in', 4751), ('of', 5926), ('for', 1272), ('on', 1680)]`, respectively.

We can observed that the data distribution of the two datasets was not exactly the same, but very similar. Since our final goal is to improve the accuracy of all samples, we do not need to do over-sampling.

3. Train Bert from scratch

3.1 Generate the data[2]

Considering the lack of corpus text, using the whole word list will lead to the sparse distribution of samples in probability space, which will further result in the generation of over-fitting. I used a BPE tokenizer to reduce the number of words.

3.2 How to train[3],[4],[5]

Like traditional Bert, we use MaskLM for unsupervised training. To speed up the training, I put the data through `. ; ,` 's prioritized to short sentences of maximum length `128`.

Pretrain with MaskLM

I tried several hyperparameters and finally found that `lr=5e-4; batch_size=64; warmup_steps=10000` is the most favorable for the loss function to decline, and the following is the result.

train steps	loss function	accuracy in valid set
20000	3.303300	0.6119
30000	2.912500	0.6381
40000	2.650100	0.6076

We can simply observe that after 30,000 steps, the performance of the model starts to get worse and worse, indicating that the data set is small enough to be remembered by the model and it loses its generalization ability. In short, it is not large enough to prevent overfitting.

Finetuning with remain set

We used checkpoint-30000 for finetuning, but the effect was not satisfactory: when we fixed Bert's parameters, using valid set as train set only brought 0.6 accuracy.

3.3 Some possible further explorations

Here are some possible ways to improve the overfit. Due to the lack of time, I didn't try any further.

Regularization

Modify the L2 regularized hyperparameters of the model to achieve a balance between overfitting and unlearning.

Data argument

Used the trained Bert model to add and modify some data and expand the size of the training set.

4. N-gram with Random forest

4.1 Generate the feature by Bi-directional N-gram

We use `class Ngram()` to count all n-tuples, and the end result is that we have a dict for all k-tuples, take n=5 to balance the amount of calculation and accuracy. To make sure that strings of less than n lengths can be counted, we add two padding lengths of n on each side of the string.

In order to better count the semantic information from the front and the back, we do n-gram classification for both directions. For each K-Gram classifier, we output features with a length of 5, representing the probability of each category respectively. Due to `k=1..n`, and we have forward and backward direction, the number of total feature dimensions is `2*5*5=50`.

For each K-Gram classifier, we do use label smoothing and normalization.(Because n gram is used to text generation but we only need classification.) And the final value is:

$$\text{Set } occur(x) \text{ as the occurance times in the Training text.} \tag{1}$$

$$P_i(X) = \frac{occur(i) + 0.1}{\sum_k occur(k) + 0.1}$$

4.2 Naive training in Whole train set.[6]

We simply use random forest, a learning method that performs better when features are more regular. And extract the feature for the Whole train set. However the accuracy is 0.59.

In fact, after generating the training data with train Data, the text of trainset and validset is not identically distributed: In probability space, the occurrence probability of 5-gram is very low, but since trainset is used to generate 5-gram, we can generally observe this phenomenon in 5-gram, whereas in validset, this phenomenon is not observed.

So we considered two solutions: divide part of the validSet for training, or divide part of the Trainset for not constructing the n-Gram.

4.3 Split the validset to train

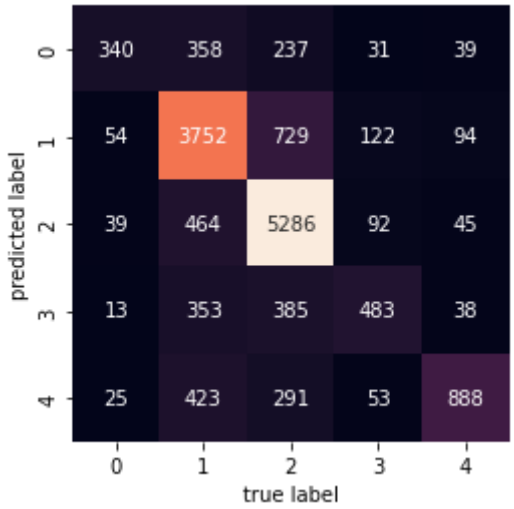
We randomly split the Validset into training set and test set, and then run the code.

	precision	recall	f1-score	support
0	0.36	0.61	0.45	144
1	0.82	0.71	0.76	1377
2	0.87	0.81	0.84	1613
3	0.40	0.57	0.47	220
4	0.58	0.76	0.66	305
accuracy			0.75	3659
macro avg	0.60	0.69	0.64	3659
weighted avg	0.78	0.75	0.76	3659

4.4 Split the trainset and test on valid set

We used the last 2200 lines of trainset as the data of training random forest, the results are as follows:

	precision	recall	f1-score	support
0	0.3383	0.7219	0.4607	471
1	0.7897	0.7013	0.7429	5350
2	0.8920	0.7630	0.8225	6928
3	0.3797	0.6184	0.4705	781
4	0.5286	0.8043	0.6379	1104
accuracy			0.7345	14634
macro avg	0.5857	0.7218	0.6269	14634
weighted avg	0.7820	0.7345	0.7490	14634



Since we are not allowed to train with validSet, the final answer will be generated by this model.

5. Reference

[1] https://openaccess.thecvf.com/content_ICCV_2019/papers/He_Rethinking_ImageNet_Pre-Training_ICCV_2019_paper.pdf

[2] https://huggingface.co/docs/tokenizers/python/latest/quicktour.html

[3] https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/token_classification.ipynb#scrollTo=imY1oC3SIrJf

[4] https://colab.research.google.com/github/huggingface/blog/blob/master/notebooks/01_how_to_train.ipynb#scrollTo=G-kkz81OY6xH

[5] https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/token_classification.ipynb#scrollTo=imY1oC3SIrJf

[6] https://blog.csdn.net/jasonzhoujx/article/details/81911799

Appendix

A. Generate `test.fo2` from huggingface pretrained bert

The notebooks `COMP3361Assignment1GenerateTestData.ipynb` is a simple example of generating labels from Bert.

Although I directly used Bert to obtain a label with high accuracy, I did not use it for fitting when training my model.

I also shared it with `Xijia Tao`, `Jiayi Xin`