MSBD6000H Project 1: Argument Impact Classification

JIANG Yuxin
20710998

HUANG Yitian
20718885

KATTEKOMMULA Niharika Reddy
20729573

CHEN Yingyan
20711904

Abstract

This paper presents our submitted system to the project: Argument Impact Classification. Our team (group 01, *Bayes is all you need*) finally achieves 62.439% macro F1 score on the leaderboard, which is ranked 2 among all groups. The display names of our kaggle accounts are *JIANG Yuxin*, *ustust*, *Niharika Reddy Kattekommula and YYannisChan*.

1 Dataset

Here we briefly introduce the dataset we use. Given a context which is a path from the root to the parent of text in the argument tree, we name the nearest member of context to the text as *parent 1*, the second nearest member of context to the text as *parent 2* (if exists) and so on. Intuitively the impact label of a text is not only related to itself but also highly depends on its context, due to the reason that the argument path is written in consideration with the line of reasoning so far. As a result, we want to see if the model performance is affected by the input number of context members, which we call it context length in the following parts. Some statistics are shown in Table 1, and we could observe that more than half of examples have 3 or higher context length.

Besides, we also regard stance label as an import feature for models to determine the label, and we name it with the same method as the context (from *stance label 1* to *stance label n* according to the distance from the text). Here we ignore "*NULL*" because it has no meaning.

Context length	Number of examples
1	1,524
2	1,977
3	1,181
4	731
5	605
>5	1,368

Table 1: Statistics of context length of the whole dataset.

2 Algorithms

We use pre-trained language models (PrLMs) such as BERT (Devlin et al., 2019) as the encoder part of our algorithm, which is depicted in Figure 1. Suppose the input context length is set as n, after tokenization, we concatenate *stance label 1* and *parent 1*, *stance label 2* and *parent 2*, ..., *stance label n* and *parent n* with text, separated by a special [SEP] token. Besides, we also add a [CLS] at the beginning and a [SEP] at the end of the sequence. Then we feed these inputs through our PrLM in order to learn better representations. Finally, we put the output embedding of the [CLS] token into a linear classifier followed by softmax to make predictions.

Weighted Loss Note that the proportion of three categories of impact label (NOT_IMPACTFUL, MEDIUM_IMPACT and IMPACTFUL) is around 1: 1: 3 which is unbalanced, so we replace the standard cross-entropy with the weighted cross-entropy by adding weights to the corresponding categories,

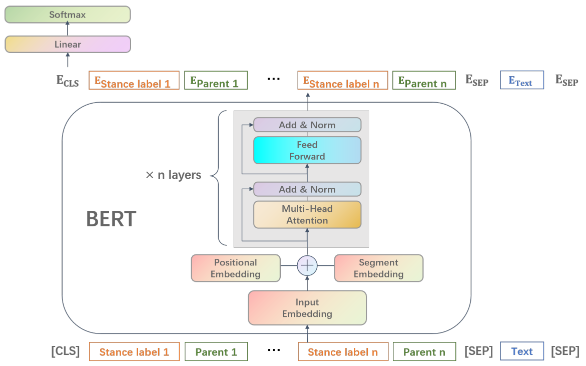

Figure 1: The overall model architecture.

Model	Macro F1
$\text{BERT}_{\text{base}}$ + C1	55.70
$\text{BERT}_{\text{base}}$ + weighted loss + C1	56.86
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C1	58.78
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C2	58.06
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C3	57.93
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C1 + S1	58.11
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C2 + S2	59.55
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C3 + S3	60.43
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C4 + S4	61.25
$\text{BERT}_{\text{base}}$ + weighted loss + label smoothing + C5 + S5	**61.74**
$\text{BERT}_{\text{large}}$ + weighted loss + label smoothing + C5 + S5	58.98
$\text{DeBERTa}_{\text{base}}$ + weighted loss + label smoothing + C5 + S5	58.40
$\text{ALBERT}_{\text{base}}$ + weighted loss + label smoothing + C5 + S5	58.56
$\text{DistilBERT}_{\text{base}}$ + weighted loss + label smoothing + C5 + S5	60.90

Table 2: Model comparison on valid dataset. The number behind C and S means the input length of context and the input length of stance label respectively.

3.0, 3.0, 1.0 respectively.

Label Smoothing Label smoothing is a well-known "trick" to improve the model's performance effectively, which encourages the activations of the penultimate layer to be close to the template of the correct class and equally distant to the templates of the incorrect classes (Müller et al., 2019). The original cross-entropy only focuses on whether the positive example is true and does not pay attention to the negative examples' relationship. In contrast, the cross-entropy with label smoothing for a single example is as follows:

$$L = -\sum_{k=1}^{K} y_k log(p_k) \tag{1}$$

$$y_k = \begin{cases} 1-\varepsilon, & right\ answer \\ \frac{\varepsilon}{K-1}, & wrong\ answer \end{cases} \tag{2}$$

where K is the number of categories. We set ε as 0.1 in our models.

3 Experimental Results and Analyses

The settings of environment and hyperparameters are listed in A and B, which can be used for reproducing our results.

3.1 Model Comparison

Table 2 shows the comparisons between different approaches. We mainly choose $\text{BERT}_{\text{base}}$ as the PrLM of our architecture described in Section 2. From the first three lines we can see that utilizing weighted loss and label smoothing really helps to improve the performance. However, we found that by only adding more context, the macro F1 score decreases. On the contrary, increasing the input length of context and stance label simultaneously can steadily improve the performance of the model, which implies the necessity of stance labels. With the input length of context and stance label equal to 5, we acquire the best macro F1 score (61.74%) on valid set, which is only slightly better than input length equal to 4, so we guess adding more context and stance label will not raise the F1 score dramatically.

In addition, we also make comparison by using various PrLMs. From Table 2 we could see that more sophisticated PrLMs like ALBERT (Lan et al., 2020), DistilBERT (Sanh et al., 2019) and DeBERTa (He et al., 2020) do not achieve better results. Increasing the model size also impairs the performance. We think the reason may be that more complex models are easily to be overfitting on the valid set under the premise that the training data is small.

After the model (we pick the model with the highest F1 score on valid set) is well trained, we extract the output embedding of [CLS] token of each example, and utilize Principal Component Analysis (PCA) to do dimension reduction as well as visualization. From Figure 2 we can observe that the embeddings of training examples belonging to different categories are well seperated, but these of valid set are mixed together, especially the

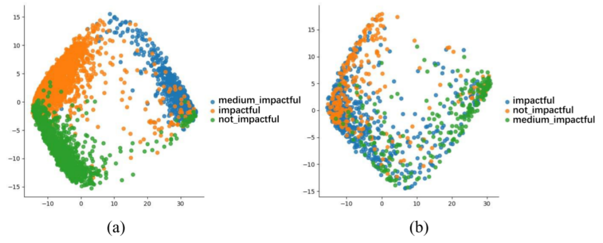

Figure 2: Visualized output embeddings of train (a) and valid (b) set after doing PCA. The explained variance ratios of train and valid set are [0.75 0.11] and [0.81 0.09], separately.

”NOT_IMPACTFUL” ones and the ”IMPACTFUL” ones. That explains why the model is overfitting on valid set.

3.2 Further Improvement

After submitting our predicted result to kaggle, we found that F1 score on test set is sometimes much lower than that on valid set. We guess it may because the learned distributions of representations of valid and test set become more inconsistent after more training steps. Here we propose a method based on KL-divergence to narrowing the gap between the valid and test set. We extract the output embeddings of [CLS] token of valid and test set and split each dimension (for example, the output embedding of $\mathrm{BERT}_{\mathrm{base}}$ has 768 demensions) into several bins. Then the probability of each bin could be computed by the number of examples fall in this bin devided by the number of all examples. Consequently, the KL-divergence between valid and test set is calculated by this formula:

$$D_{KL}(T||V) = \sum_{i}^{N} T(i)log\frac{T(i)}{V(i)} \qquad (3)$$

where T and V represent test and valid set, N is the number of bins (here we choose 100), T_i and V_i means the probability of the i_{th} bin of test and valid set.

While training, we visualize the dynamic changes of Macro F1 on valid set and KL-divergence between valid and test set over number of training steps, as depicted in Figure 3. It is clearly seen that although the F1 score on valid set increase with more training steps, the KL-divergence between test and valid set also rises, which means we may not get ideal score on test set. By observation, we apply a new stragety to stabilize our model: during training, we save the

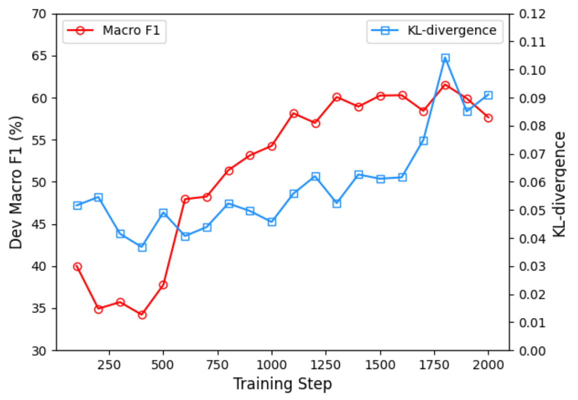

Figure 3: Macro F1 on valid set and KL-divergence between valid and test set over number of training steps.

model not only has better macro F1 but has the KL-divergence less than 0.06. Finally, we ensemble 5 single models with different parameters based on a majority vote scheme to make the final submission.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4696–4705.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

A Environment Settings

All of our codes are written based on PyTorch Transformers [1]. The third-party libraries we use are: os, time, logging, random, tqdm, numpy, pandas, scipy, ast, contextlib, argparse, torch, transformers, sklearn, matplotlib and seaborn.

B Hyperparameter Settings

The maximum sequence length is set to 384. We choose mini-batch size equal to 16 and the AdamW optimizer with an initial learning rate of 2e-05. We use some strategies for more stable training: 1) clip the gradient norm to 1; 2) adopt a linear scheduler with warm up of the first 10% training steps. We trained all the models for 10 eopchs, evaluate on the valid set at every 100 training steps and save the model with better macro F1 as well as the KL-divergence less than 0.06. For each single model, we run experiments for 5 times with different random seeds and use the average as the ultimate performance.

[1] https://github.com/huggingface/transformers