Project 1 Report: Argument Impact Classification

Chapter 1. Group Information

- Kaggle Team Name: Yellow Submarine
- Group Number: 14
- Names & Student IDs & Kaggle Display Name of Team Members:
    - YUAN Yanzhe: 20728555 (bighead)
    - LIU Donghua: 20731100 (Danhuang)
    - SHEN Dinghui: 20726478 (mengnan)
- Final Rank and Scores:
    - Score: 0.59839, Rank: 4

Chapter 2. Algorithm Description

1. Task Description

The project is about argument impact classification. The training data contains five segments as well as the validating data: [id, text, context, stance_label, and impact_label]. The testing data has the same columns except for the impact_label. So, the task is to use the information in the argument tree consisting of arguments and stances to predict the impact level of a text. In other words, we should train a 3-label classifier to achieve a best Macro-F1 performance.

2. Algorithm Introduction

Basically, we use BERT for this project. BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text. The same pre-trained model parameters are used to initialize models for different downstream NLP tasks like Question Answering, Text Classification, etc.

3. Algorithm Description

Here is the detailed description of our algorithm.

3.1. Data Preprocessing

Data preprocessing is an important step for NLP tasks. In this project, we use re package to filter useless content in the raw text like '[]' and transform symbols to words like '$' to 'dollar'. Next, we transform the impact_label to {0,1,2}, which corresponds to {NOT_IMPACTFUL, MEDIUM_IMPACT, IMPACTFUL}. Finally, we gather the data for the next step.

3.2. Feature Engineering

After preprocessing our data, we plan to further construct our features to better fit our models. We take the following methods to form our data for the model.

3.2.1. Combine the Text and the Context as the Input

Since the text is correlated with the context, whether the context is convincing or not decides the impact of the text. So, we believe that text and the context (as a whole) should be a sentence pair as our features. In this way, we prepare [text, context, impact_label] for our training and validating data.

3.2.2. Match the Stance Label to the Corresponding Sentence

After detailed studying the examples given in the project task introduction document, we came to the conclusion that the first sentence of the “context” is the root node of the Argument tree, which represents the thesis, corresponding to the “NULL” in the “stance_label”. In addition, the label corresponding to the first sentence of “text” is the last label word in “stance_label” (Support/Oppose).

With this finding, to further provide the useful information contained in "stance_label" to our model, we spliced the words in "stance_label" to the end of the corresponding sentence.

> *<text>: There is very little verifiable evidence to state that anyone has been deterred from serious crimes by danger of torture in the future. For a 2014 study, see mdpi.com 'OPPOSE'*
>
> *<context>: Physical torture of prisoners is an acceptable interrogation tool.'NULL' Torture, conceptually, could act as an excellent deterrent; knowing torture is acceptable is likely to make individuals reconsider actions that leave them vulnerable to it. 'SUPPORT'*

Figure 1. An example of processed data

3.3. BERT Fine-Tuning

In this part, we use PyTorch and Huggingface's Transformer to build a BERT classifier, fine-tune it on the training data and perform impact predicting on the testing data.

3.3.1. Data Loader

Before defining our classifier, we should turn our data into forms that can be accepted by BERT. We use

BertTokenizer to tokenize our text data. BertTokenizer can help split sentences into words and build up the vocabulary so that the meaning of text can be easier interpreted in the form of token sequences. In addition, it is also good at handling out-of-vocabulary words. So, we use encode_plus function to encode [text, context] in the form of '[CLS] text [SEP] context [SEP]'. This function helps tokenize our sentences, add special [CLS] and [SEP] tokens, truncate the concatenated sequence to max_length and create attention masks. Then, we put the data into PyTorch DataLoader, which creates an iterator for the dataset and helps in saving memory and boosting speed.

3.3.2. BERT Classifier

After that, we construct our BERT classifier. The classifier contains the following parts:

- BERT model
- Dropout
- Linear layer

When the data in the form of [input_ids, attention_masks] approaches, the BERT model outputs the result. Next, the last_hidden_state, i.e. the [CLS] token in the last BERT layer is selected from the result as the sentence representation. Then a dropout layer and a linear layer is followed. The dropout is used to avoid overfitting and the linear layer is used to reduce the feature dimensions to 3 for classification.

```
class BertClassifier(nn.Module):
    def __init__(self, pretrained_model, num_class):
        super(BertClassifier, self).__init__()
        self.bert = pretrained_model    #BertModel.from_pretrained('bert-base-uncased')
        self.dropout = nn.Dropout(p=0.5)
        self.linear = nn.Sequential(
                                    nn.Linear(self.bert.config.hidden_size, 128),
                                    nn.ReLU(),
                                    nn.Dropout(p=0.5),
                                    nn.Linear(128, num_class)
        )

    def forward(self, input_ids, attention_mask):
        res = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        res = self.dropout(res['last_hidden_state'][:,0,:])    # res['last_hidden_state'][:,0,:], res['pooler_out
        res = self.linear(res)
        return res

# settings
net = BertClassifier(bert_model, 3).to(device)
optimizor = AdamW(net.parameters(), lr=2e-5, correct_bias=False)
```

Figure 2. Implementation of the model based on Pytorch

Besides, in our model, AdamW is used as the optimizer, and CrossEntropyLoss is used as the loss function. Moreover, we use the transformer's get_linear_schedule_with_warmup method to schedule the learning rate so that the convergence process can be faster and more stable.

4. Train, Evaluate and Test the Model

For every epoch, we first train our model on the training data, and then evaluate our model on the validating data based on criteria like Macro-F1, loss, and accuracy. During the process, we record the model with the best F1 score or cross entropy loss on the validating data. In the end, we use the best model to predict impact_label on the testing data.

Chapter 3. Hyperparameter Tuning

1. Parameter Tuning

Here are the hyperparameters and the range we chose for tuning:

- max_length: [128:512]
- dropout: [0.4:0.8]
- AdamW learning rate: [1e-5:5e-5]

As for these parameters, we remain the same:

- batch_size: 32
- epoch: 6

In addition, we have also tried the following comparison experiments:

- try other model in BERT family: DistilBERT
- try different NN structure after BERT: use [dropout, linear, dropout, linear] after BERT output
- try different output from BERT: use ‘pooler_output’ instead of ‘last_hidden_state’

All in all, the best model we have is with these hyperparameters:

- {max_length: 256, dropout: 0.5, Adamw lr: 2e-5, batch_size: 32, epoch: 6}

2. Difference Between Local Validation and Online Result

During the experiment, we found that there will always be some differences between our local verification results and the submitted scores. In fact, the hyperparameters and model structure that provided the best local result did not score particularly high in the submission system. We believe that the reason for this phenomenon is that the training set and test set used in the experiment are not too large, and the model that performs well on the validation set may not fit the test set data best.

Chapter 4. Other Information

1. How to Run:

- Using Google Colab to run the corresponding .ipynb file

2. Third-party Libraries Dependency

- transformers, nltk, pytorch, sklearn, matplotlib, pandas, numpy.

REFERENCE:

[1] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805

[2] Durmus, Esin and Ladhak, Faisal and Cardie, Claire. 2020. The role of pragmatic and discourse context in determining argument impact. arXiv preprint arXiv:2004.03034

[3] https://huggingface.co/transformers