Sequence Labelling

Text Mining Assignment 2

Zhipei Qin s3977226 Peiwen Xing s3838501

1 Introduction

This assignment is to train an NER classifier on the W_NUT17 data set for the "Emerging and Rare entity recognition" task. We begin by converting the data which is suitable for token classification; The tokenizer are downloaded and cached using BERT pre-trained models. We set up the evaluation for the data set, and get the baseline results with default hyperparameter settings and results with optimized hyperparameter settings. The results after hyperparameter optimization for the different entity types are also obtained.

2 Methods

2.1 Task and Data sets

For this task, we use the *wnut_17* data sets to mainly detect and classify novel entities in noisy text. There are six classes of entity in this task: **Person**, **Location** (including GPE, facility), **Corporation**, **Product** (tangible goods, or well-defined services), **Creative work** (song, movie, book, and so on), and **Group** (subsuming music band, sports team, and non-corporate organizations). The *wnut_17* contains three IOB data sets: a train set (3394 rows), a validation set (1009 rows) and a test set (1287 rows). Each data set contains three features: **id**: ID of the example text; **tokens**: Tokens of the example text; **ner_tags** : NER tags of the tokens (using IOB2 format), with possible values: 0:O; 1: B-corporation; 2: I-corporation; 3: B-creative-work; 4: I-creative-work; 5: B-group; 6: I-group; 7: B-location; 8: I-location; 9: B-person; 10: I-person; 11: B-product; 12: I-product.

2.2 Data processing

We first pre-process existing annotated text data into the data structure for token classification in Huggingface and align the labels with the tokens. By defining two functions *align_labels_with_tokens* and *tokenize_and_align_labels*, we use a tokenizer to tokenize the text sequence and align the text sequence with its corresponding label. For the special tokens added by the tokenizer ([CLS] at the beginning and [SEP] at the end), set the label of all these tokens to -100 so that the corresponding predictions are ignored in the loss calculation. Through *DataCollatorWithPadding* of the $Transformers$, we define a collate function that pads the length of all samples in each batch to the maximum length within the batch.

2.3 Model Building and fine-tuning

We begin by setting up label mappings, mapping numeric identifiers to label names and vice versa. We then load a pre-trained model suitable for sequence labeling tasks, such as "bert-base-cased" using the *AutoModelForTokenClassification* class from the Huggingface library, providing the label mappings to the model. Next, we create a *TrainingArguments* object to specify the model's save path and seed to ensure reproducibility of the results. With the *Trainer* object, we conduct model training, utilizing training and test datasets, data collation, performance evaluation functions, and tokenization.

2.4 Hyperparameter optimization

First, we train a model with the default hyperparameter settings and evaluate the model on the test set to obtain baseline results. Next, we define multiple sets of hyperparameters for optimization, including the following specific values:
Learning rates: [2e-5, 3e-5, 4e-5]
Batch sizes: [8, 16]
In a nested loop, we explore different hyperparameter combinations, training the model and evaluating its performance on the dev set. Finally, we find the best hyperparameter combination and evaluate the model on the test set. We tried hyperparameter optimization both with and without the AdamW optimizer and found that the best learning rate and batch combinations were both 0.00004 and 16.

2.5 Evaluation

We use the *seqeval* framework to evaluate token classification prediction. By designing the *compute_metrics*() function, we can return the overall Precision, Recall, and F1-score on the test set. Furthermore, the evaluation function is extended to compute the Precision, Recall, F1-scores, and the macro- and micro-average F1 scores for all entity types (for B-label, I-label, and the full entities).

3 Results

The baseline results and the results after hyperparameter optimization as shown in the table below.

	Precision	Recall	F-score
Baseline	0.5254	0.3068	0.3874
Hyperparameter optimization without AdamW optimizer	0.5322	0.3216	0.4009
Hyperparameter optimization with AdamW optimizer	0.2873	0.4740	0.3578

4 Discussion and Conclusion

4.1 Question 1

Setting the learning rate and batch size to the optimal combination of 0.00004 and 16. We can found that there is an improvement in precision, recall, and F-score compared to the baseline results without using the AdamW optimizer. However, when using the AdamW optimizer, Recall improves relative to the baseline results, but both precision and F-score decrease.