NER and Classification with LLMs

Len Brandes

November 2024

1 Introduction

In this report, I discuss the results of finetuning a large language model (LLM) of the BERT architecture for the tasks of named entity recognition (NER) and sentence classification based on the CONLL dataset. First, the model is trained for each task individually and then to do both tasks simultaneously using a single forward pass. The performance of the individual models versus the multitask model is discussed.

2 Experiments

2.1 Token classification

For the first experiment, I have trained a pretrained BERT model with a model head for token classification on 1000 random examples of the CONLL dataset, which are already labeled for named entity recognition. The distribution of token labels in the training set is illustrated in Figure 1. Most of the tokens in the dataset are labeled as '0', i.e., they do not correspond to any specific entity.

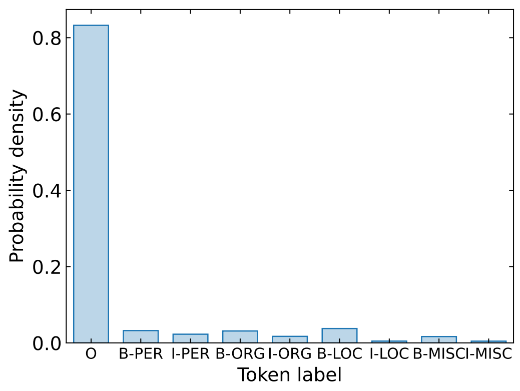

Figure 1: Distribution of token labels in the training data.

After training the model for 4 epochs with the Adam optimizer, a learning rate of 5e-5 and a weight decay value of 0.01, it displayed the following performance on a test set:

$$F_1 = 84.3\% \qquad \text{Accuracy} = 95.4\% \ . \tag{1}$$

The learning curves of the F_1-score and the accuracy as a function of training examples are displayed in Figure 2.

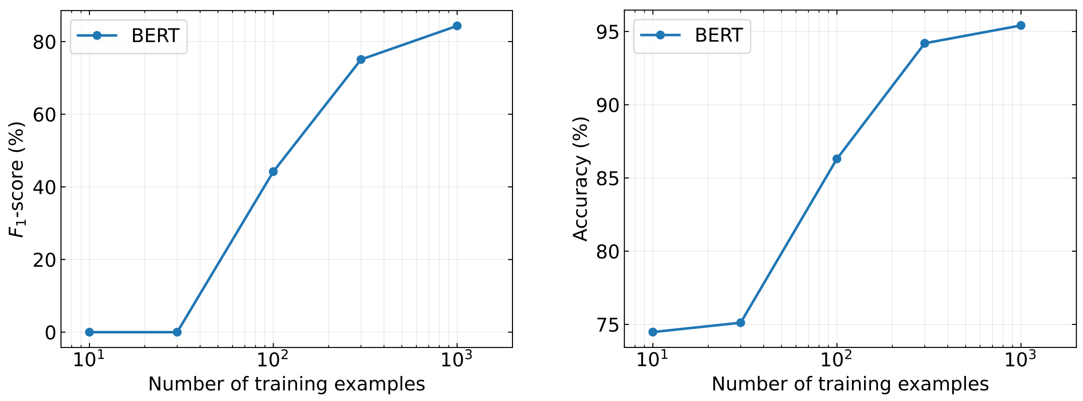

Figure 2: F_1-score (left) and accuracy (right) of a BERT model finetuned for named entity recognition as a function of the size of the training set.

The F_1-score is zero for a training set consisting of only 10 examples. In contrast to the accuracy, it also does not increase when the size of the training set is increased to 30. Due to the huge amount of tokens without entity, see Figure 1, for a small training set the network only learns to classify most of the tokens as '0'. However, further increasing the training set to size of 100 leads to a significant improvement in both accuracy and F_1-score. This increases starts to slow down towards training set sizes of 1000 examples.

2.2 Sequence classification

In the next experiment, I have augmented the training data with sequence-level labels depending on their topic: (0: World, 1: Sport, 2: Business, 3: Technology, 4: Other). For that I used the API of the GPT-4o mini model. After concatenating the token labels of the CONLL dataset to sequences, I labeled them using the prompt in Figure 3.

Here, I set the temperature of the GPT model to 0.2 to reduce the amount of randomness in its outputs. Looking at a few random examples in Table 1, the labeling of the sequences seems reasonable:

```
Task: Classify the given document based on its topic into
one of the following categories:
0: World
1: Sport
2: Business
3: Technology
4: Other
Instructions: Analyze the content of the input sentence and
assign the appropriate topic label as an integer (0 to 4).
Output Format: A single integer (0, 1, 2, 3, or 4) represent-
ing the most relevant topic category.
Input: [INPUT SENTENCE]
```

Figure 3: Prompt used to augment the dataset using the API of GPT-4o mini.

Sentence	Label
‘*The Norwegian news agency NTB quoted another official on the island as saying no survivors had been found.*’	World
‘*Slask Wroclaw 3 Odra Wodzislaw 1*’	Sport
‘*Lamonts said it plans to issue 9 million shares of new common stock.*’	Business

Table 1: Random sequences from the CONLL dataset labeled using the API of GPT-4o mini with the prompt in Figure 3.

The resulting distribution of classes of the sequences in the training data is displayed in Figure 4. There are only very few sequences focused on technology in the dataset. Moreover, a large proportion of the sentences are labeled as ’Other’, because the contain only dates and places, e.g., ‘*WINNIPEG 1996-08-26*’. The training could be improved by increasing the number of sequences labeled as ‘Technology’ or using only sequences with a minimum number of tokens. The prompt in Figure 3 could be improved by proving few-shot examples created by hand. However, I found that this leads to only a small difference in the resulting labels.

Training a classifier based on a pretrained BERT model with a sequence classification head using 1000 training examples and the same hyperparameters as in the previous experiment yields the following test performance:

$$F_1 = 87.0\% \qquad \text{Accuracy} = 86.9\% \ . \tag{2}$$

The confusion matrix for the class labels is depicted in Figure 5. There are too few samples in the test dataset to evaluate the performance for the ‘Technology’ class. The labels ‘Sport’, ‘Business’ and ‘World’ all have non-vanishing prediction overlap with ‘Other’. Moreover, there appears to be some finite overlap between ‘World’ and ‘Business’.

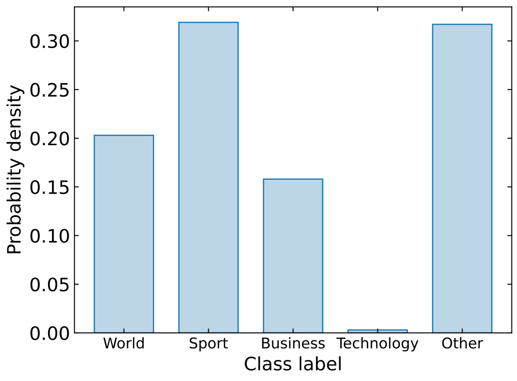

Figure 4: Distribution of sequence-level class labels of the training data augmented using GPT-4o mini.

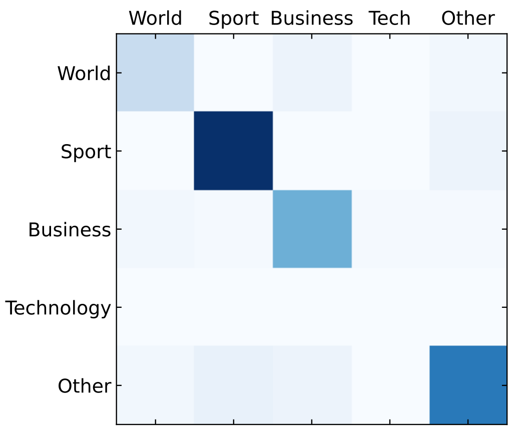

Figure 5: Confusion matrix for the sequence classification task.

2.3 Multitask

Finally, I trained a single BERT model to perform both tasks, token- and sequence classification, with a single forward pass. For this, I put two separate task-specific heads on top the contextual embedding of the input tokens provided by BERT. For the two model heads I followed the architectures used in the previous experiments; the sequence classification head takes the hidden-state of the first token, i.e., the classification token, which is processed through a linear layer with an tanh activation. This output is then passed to another linear layer resulting in 5 outputs for the five different sequence classes.

The token classification head takes all token embeddings from the encoder output and passes each embedding through a shared feed-forward network to predict the token-level label. For joint training I used a combined loss function consisting of the cross-entropy losses $\mathcal{L}_{\text{sequence}}$ and $\mathcal{L}_{\text{token}}$ for sequence-level and token-level labels, respectively, weighted with parameters α and β:

$$\mathcal{L}_{\text{tot}} = \alpha\mathcal{L}_{\text{sequence}} + \beta\mathcal{L}_{\text{token}} \quad . \tag{3}$$

Empirically I found that a higher weighting of the token loss compared to the sequence loss, using, e.g., a rate of 1 to 7, lead to a better performance. Moreover, the combined model required a significantly larger amount of epochs for training. In principle, it would also be possible to learn the parameters α and β also during training. The resulting F_1-scores and accuracies of the multitask model for both token and sequence classification are depicted in Table 2 and compared to the two singletask models used in the previous experiments.

	Token classification		Sequence classification	
Model	F_1-score	Accuracy	F_1-score	Accuracy
Singletask 1	84.3%	95.4%		
Singletask 2			**87.0%**	86.9%
Multitask	**85.0%**	**96.6%**	**87.0%**	**87.0%**

Table 2: Performance of the multitask model for token and sequence classification compared to the two singletask models.

The performance of the multitask model increases slightly both for sequence as well as for token classification compared to the singletask models discussed in the previous sections. The reason for this is most likely that the information about the entities in a sequence can also be helpful in determining the subject of the sequence (and vice versa). I have checked that training the combined model with only one of the two losses, i.e., setting either α or β to zero, results in similar performances on the single tasks as before.