Multi-label text classification with RoBERTa
Project report
Group 3

Teemu Pöyhönen
014696400

Christoph Schäfer
015430555

Varpu Vehomäki
014810189

1 Introduction

For our final project we chose the text classification task. The task in question was a multi-label classification task with 126 different labels.

We implemented RoBERTa base after experimenting with other BERT models (DistilBERT, BERT base.) The main issues are the large number of labels and some of these labels being very infrequent.

2 Data

The data used for this task was from the Reuters corpus. The original data was in xml-format, so we created an xml scraper to retrieve the topic codes, headlines and text.

There were 126 different labels and 296934 instances. We split the data so that there were 237547 instances in the training set and 59387 instances in the evaluation set.

Some labels were much more common than others. The most common label was CCAT that has the description ”corporate/industrial”. There were also labels that did not appear in the training set at all. In Figure 1 we can see the differences between the numbers of labels.

3 Model

The model is based on (Mishra, 2020), which consists of fine-tuning BERT base (uncased). Afterwhich, there is a dropout layer (0.3), followed by a fully connected linear layer.

For the specific BERT model, we have tried BERT-base, BERT-large, DistilBERT, RoBERTa and XLM-RoBERTa.

3.1 Architecture

Initially, we fine-tuned and ran experiments on simply the BERT model and a linear layer on top of it, though for learning purposes, we changed the model from BERT model to BERTForSequenceClassification. This required some coding and adapting the data to the model, as well as changing the architecture. Lastly, we switched back to manually implementing the linear layer for the purpose of implementing two linear layers,and a relu activation function. Our final resulting architecture is the following:

```
l1 = transformers.RobertaModel.from_
    pretrained('roberta-base',
    output_hidden_states=False)
l2 = torch.nn.Sequential(
        torch.nn.Dropout(0.3),
        torch.nn.Linear(768, 768),
        torch.nn.ReLU(),
        torch.nn.Dropout(0.3),
        torch.nn.Linear(768, 126)
)
```

Dropout rates 0.1, 0.3 and 0.5 were experimented after the BERT and BERTForSequenceClassification. 0.3 proved to be a well-performing rate, so we applied it to DistilBERT and RoBERTa.

While we are aware that dropout should not be used before the softmax layer, we still wanted to experiment and see the effect. As expected, the model performed slightly worse, and the model reported 0% training accuracy for all epochs (while validation accuracy increased in a normal way.)

3.2 Hyperparameters

After experimentation, it seems that the model learns best with 2e-05 learning rate. Regardless of varying the batch size, this value performed the best. Learning rate of 1e-04 causes the model to not learn at all (regardless of batch size) and 2e-05 is almost equal to 1e-05.

As we are trying to classify documents, we increased the maximum sequence length from 200 to 512. In fact, this is BERT’s maximum as well.

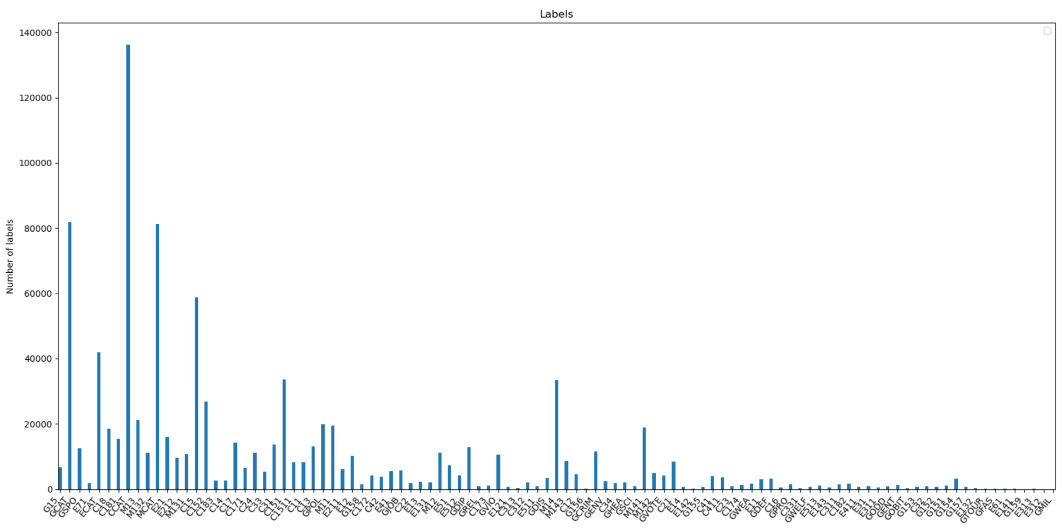

Figure 1: All labels that occured in the training set and their number of occurances

The training batch size was set to 32. Increasing this further (e.g. 64) resulted in CUDA memory errors, since higher batch size combined with the high value for maximum sequence length produces very large tensors, causing memory errors.

We trained the model for 5 epochs. It seems that on average, training accuracy surpasses the validation accuracy after approximately 4 epochs, regardless of different hyperparameters.

We use the AdamW optimizer with weight decay and epsilon.

We found from BERTForSequenceClassification model's documentation page a method to apply weight decay to all parameters other than bias and layer normalization terms. For this, weight decay 0.1 was used. This improved the validation accuracy slightly.

With so many labels (126) we implemented epsilon (1e-8) to deal with any zero division errors.

To reduce the problems caused by the class imbalance in the training set we used the function `BCEWithLogitsLoss` with the pos_weight parameter to give more weight to examples that have the label 1.

4 Training process

During the training process we recorded the training and validation accuracies to keep track of the changes over the epochs and to see if our models are overfitting or underfitting. For all the different models we implemented a similar progression of accuracies could be observed. The results of the BertModel are depicted in Figure 2, of the DistilbertModel in Figure 3 and of the RoBERTaModel in Figure 4.

As can be seen in the figures, the models achieve already good results during the first epoch of about 65-67% and increase to an accuracy of about 69-70% by Epoch 5. Even if our models are learning and the training accuracy increases over time it is still not very high and overfitting can be excluded. However, wrong predictions are still made at later epochs. This could be due to the high number of labels and especially the uneven distribution of label occurances.

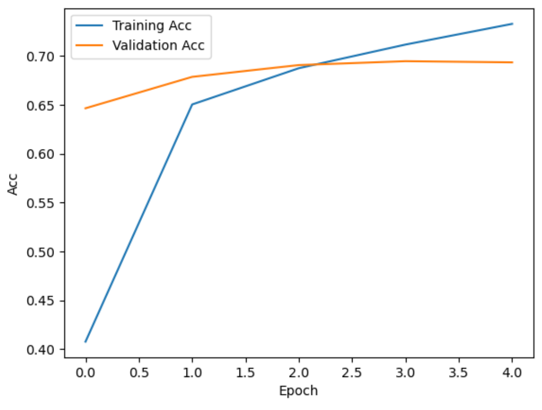

Figure 2: Progression of the training and validation accuracy of the BertModel over time

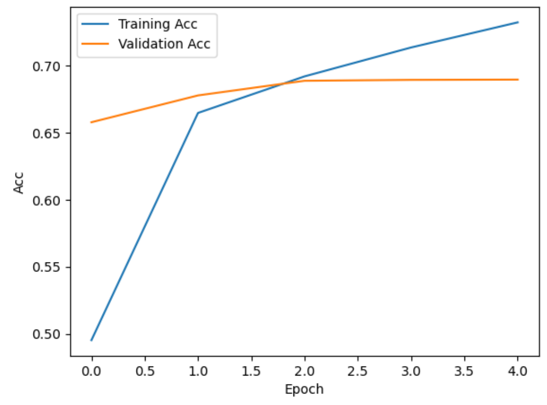

Figure 3: Course of the training and validation accuracy of the DistilbertModel over time

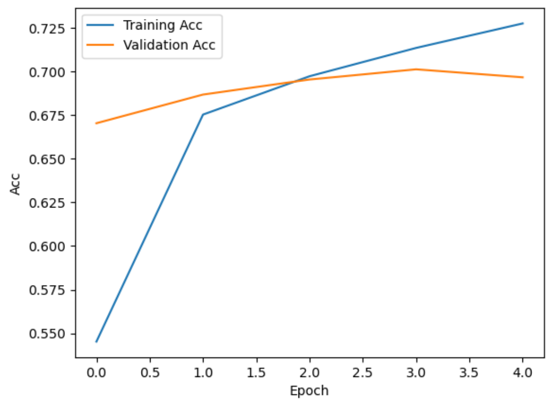

Figure 4: Course of the training and validation accuracy of the RoBERTaModel over time

5 Results

After experimenting with different models, we came to the conclusion that the RoBERTa model achieved the best scores.

Tables 1, 2 and 3 present some of the results by the RoBERTa model.

label	precision	recall	f1-score	support
GCAT	0.94	0.96	0.95	16416
C15	0.96	0.94	0.95	11668
CCAT	0.95	0.97	0.96	27288
MCAT	0.95	0.95	0.95	16132

Table 1: Most common labels

In tables 1 and 2 we can see that all of the most common labels have F1-scores over 0.9 whereas none of the least common labels gets too close to that number. Compared to some of the worse-performing models we experimented with, the precision of the least common labels is lower. In turn, the F1-scores of many of the least common labels were higher than for example with the plain BERT model.

label	precision	recall	f1-score	support
C313	0.61	0.55	0.58	74
C331	0.78	0.77	0.77	86
E132	0.78	0.84	0.81	74
E141	0.59	0.59	0.59	29
E142	0.71	0.42	0.53	12
E143	0.78	0.78	0.78	97
E312	0.00	0.00	0.00	1
E313	0.00	0.00	0.00	8
E61	0.43	0.76	0.55	17
G156	0.00	0.00	0.00	17
GFAS	0.47	0.54	0.51	35
GOBIT	0.76	0.71	0.73	62
GTOUR	0.90	0.65	0.75	40

Table 2: Least common labels

label	precision	recall	f1-score	support
micro avg	0.90	0.89	0.89	190321
macro avg	0.62	0.60	0.61	190321
weighted avg	0.90	0.89	0.89	190321
samples avg	0.92	0.91	0.90	190321

Table 3: Scores

The macro and micro averages of the F1-score seen in table 3 support the idea that the model performance when classifying the most common labels is good, but it does not handle less common labels very well. Specifically, this is evidenced by the fact that macro average F1-score is significantly lower than micro.

Our biggest problem was mostly the model overfitting the training data. The validation accuracy typically increased between the first epochs but plateaued quickly, while the training accuracy continued to get better.

The labels that appeared only few times in the training set got worse F1-scores than the ones that appeared more often. The model's performance when it comes to these labels could have been improved by doing more data analysis and by trying different sampling methods.

6 Git Repository

All models and necessary files for this project are uploaded at the following Git repository: https://github.com/Teemursu/intro2dl/tree/master/final_project

References

Abhishek Kumar Mishra. 2020. Fine tuning transformer for multilabel text classification.