Romanian Sentence Classification Project Report

Matei Bejan, group 507

January 4, 2021

1 Machine Learning Approach

Libraries: PyTorch, sklearn, transformers, sentencepiece, numpy, pandas, matplotlib, seaborn.

For this project we tried a variety of approaches and models, mostly BERT-based.

There are two important notes regarding BERT models:

1. We have used the same training parameters for all of them, only changing the model type and tokenizers, who all have the same parameters. We will discuss these hyperparameters only for the XLM-RoBERTa model we will be presenting in order for this report to not become repetitive. For the other BERTs we will just add training statistics and commentaries if necessary.

2. Tested the following data cleaning approaches with Romanian BERT:

   - Character lowering, non-alphanumeric character removal, stopwords removal and stemming (using PorterStemmer for Romanian).
   - Character lowering, non-alphanumeric character removal, stopwords removal.
   - Character lowering, non-alphanumeric character removal.
   - Nothing.

   After running all these data cleaning variations it turned out that doing nothing to the input data helped BERT learn most, so we decided to use that approach for all BERT models.

As an additional note, we wrote our own sampling function in order to sample data with respect to the data distribution. We used 80 samples for both validation and test in order to maximize to amount of data the model is exposed to. The sampling distribution can be seen in the `sample_data` procedure.

Baseline

We used a linear model with SGD training as our baseline for comparing future deep models with. This model, trained on a tf-idf representation of the data, managed to score 81% accuracy. The model was trained on raw data.

1.1 Model Choice

The model we chose for completing this task was hugginface's pretrained XLM-RoBERTa, introduced in Unsupervised Cross-lingual Representation Learning at Scale. We chose this model over multilingual BERT or Romanian BERT due to the fact that it delivered better results with similar hyperparameters.

A note on the differences in training XLM-RoBERTa and other other BERTs is that RoBERTa requires downloading the sentencepiece library and linking a vocabulary alongside specifying the model name. The vocabulary link we used is:
https://huggingface.co/xlm-roberta-base/resolve/main/sentencepiece.bpe.model.

Regarding data processing, we only used the XLMRobertaTokenizer to turn the raw text into encoded tokens for RoBERTa. The used a max length of 128 for the tokens as this was the maximum that Colab Pro would allow without crashing the session during training due to using too much GPU memory. However, this should have enveloped all training samples, as the maximum number of tokens in a sample was 112 or 114. We also applied padding and added special BERT tokens.

1.2 Hyperparameter tuning and training

We started training with a relatively high learning rate, namely 1e-3, on 10 epochs. After seeing the results improve consistently, we decided to lower the learning rate and increase the number of epochs. This led to training on 100 epochs with a learning rate of 1e-5. We observed that the validation loss did not improve after the 35^{th} epoch, so we decided on a final number of epochs of 35 and a final learning rate of 1e-5.

Regarding learning rate scheduler, we tried three schedulers that are part of huggingface's transformers library: linear scheduler with warmup, cosine scheduler with warmup and cosine scheduler with warmup and hard restarts. Out of the three it was cosine scheduler with warmup that delivered the best results, so we chose to use that.

Since BERT models are sensible to the random seed value choice, we decided to run a couple of tests with different seeds. Through trial and testing we obtained an optimum seed value of 2000.

Due to computational constraints, we couldn't use a training batch size larger than 100. However, we settled for a training batch size of 64 as there was no noticeable difference reflected in the final accuracy.

Since we using PyTorch, we wrote our own training loop. During training we decided to use torch's cross-entropy loss because it accounts for the training data distribution. Also, each epoch we checked the model's validation loss in order to save the model with minimal loss.

2 Results

2.1 XLM-RoBERTa

Our model obtains a consistent 84-88% test accuracy and 0.5 test loss. Keep in mind that we only tested on 80 samples in order to give the model as much exposure to the data as possible.

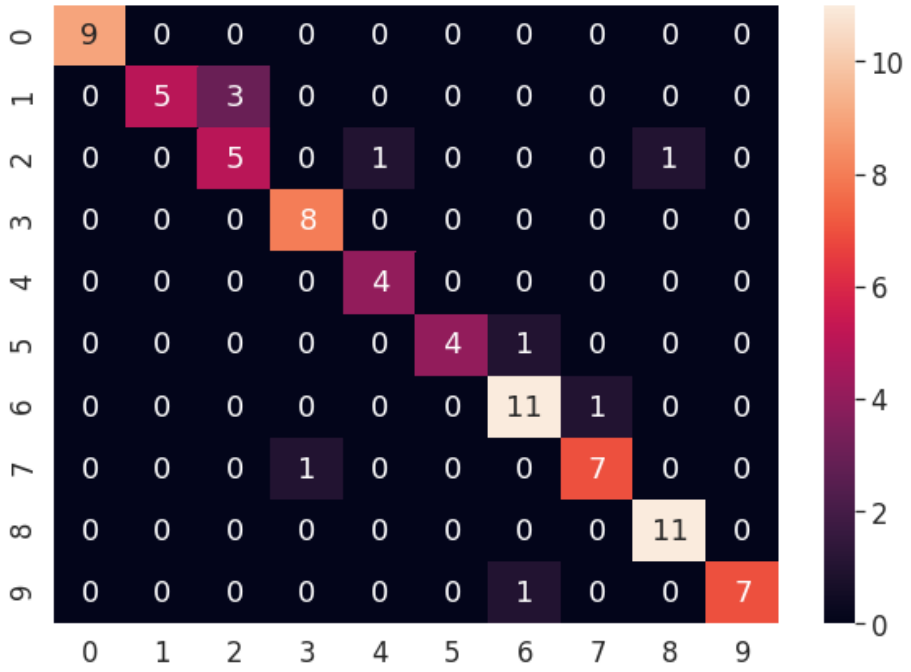

Figure 1: XLM-RoBERTa confusion matrix.

As it can be seen in the figure above, our model seems to be performing well for the 0, 3, 4, 5, 6 and 7 classes, decently on the 7 and 9 classes and poorly on the 1, and 2 classes. It also seems to confuse class 1 with 2.

We have included the training loss and accuracy in the figures on page 4.

As it can be seen in figure 2, the validation loss lowers rapidly from epochs 1 to 8, after which it registers a slight increase in value up until epoch 15, then steady but slower decrease until epoch 28 and then it stagnates.

The accuracy, however, increases steadily up until the 9^{th} epoch and then it proceeds to fluctuate around 0.75-0.78.

2.2 XLM-RoBERTa Ensemble

The model ensemble consists of 11 separate XLM-RoBERTa models (number of classes plus one). We record the predictions of each model and then we take

the most frequent prediction for each class. The process itself was very slow as we had to restart the Colab Pro environment after every training due to GPU memory restrictions.

This method seems to have boosted the test accuracy on kaggle by a 1-2 percentages, in other words from 81% accuracy to 83% accuracy.

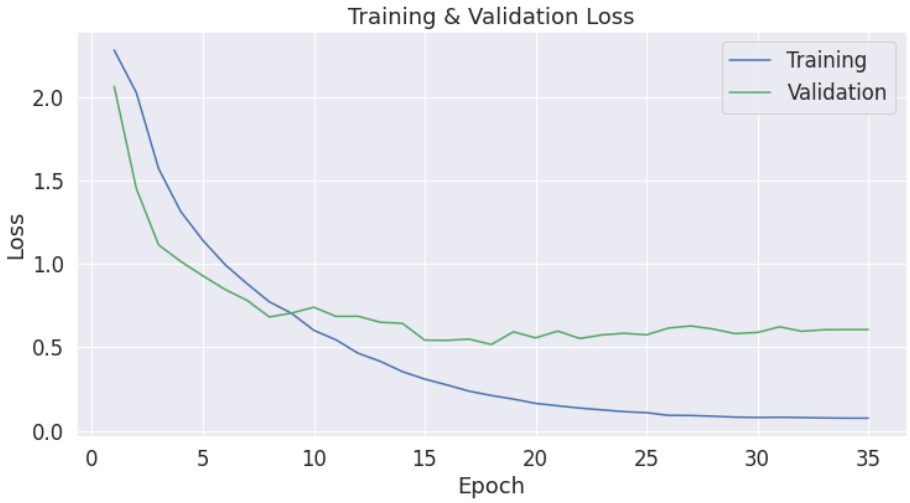

Figure 2: XLM-RoBERTa train and validation loss over 35 epochs.

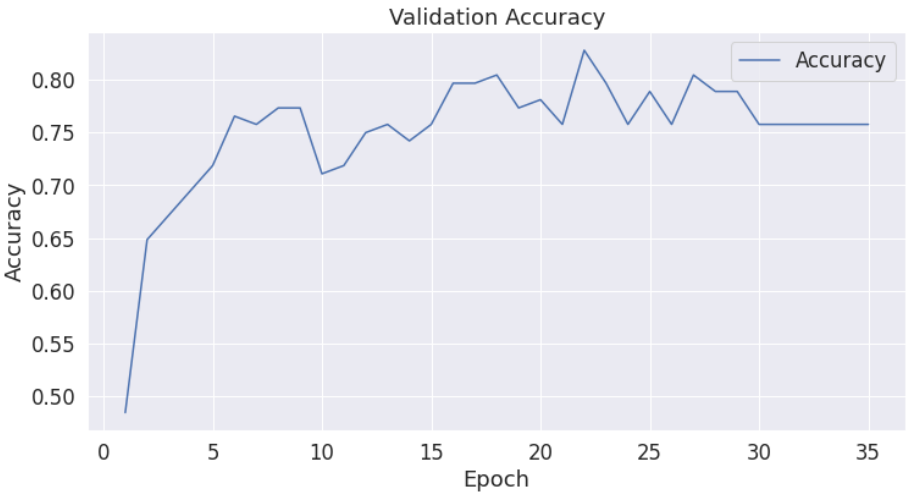

Figure 3: XLM-RoBERTa train accuracy over 35 epochs.

3 Other models

3.1 Romanian BERT

For this model we used the pretrained model created by Dumitrescu Stefan: dumitrescustefan/bert-base-romanian-cased-v1.

Romanian BERT's test accuracy was about 80%, while on kaggle it managed to score at most 75.75%, which is noticeably lower than what XLM-RoBERTa managed to get. This can be observed by comparing the two model's loss and accuracy plots. The loss plot for Romanian BERT fluctuates around 0.4 in comparison to RoBERTa's 0.5, although Romanian BERT's loss has much higher variance around 0.4. Similarly, Romanian BERT's accuracy fluctuates heavilty at the start of the training, only to stagnate after the 20^{th} epoch, with two episodes in which it is decreasing.

We can also clearly see from the confusion matrix that the model does confuses class 1 with 2, 4, and 5 and class 6 with 7 and 9.

Additionally, we have used BertTokenizer instead of XLMRobertaTokenizer as presented in **1.1 Model Choice**, but with the same parameters.

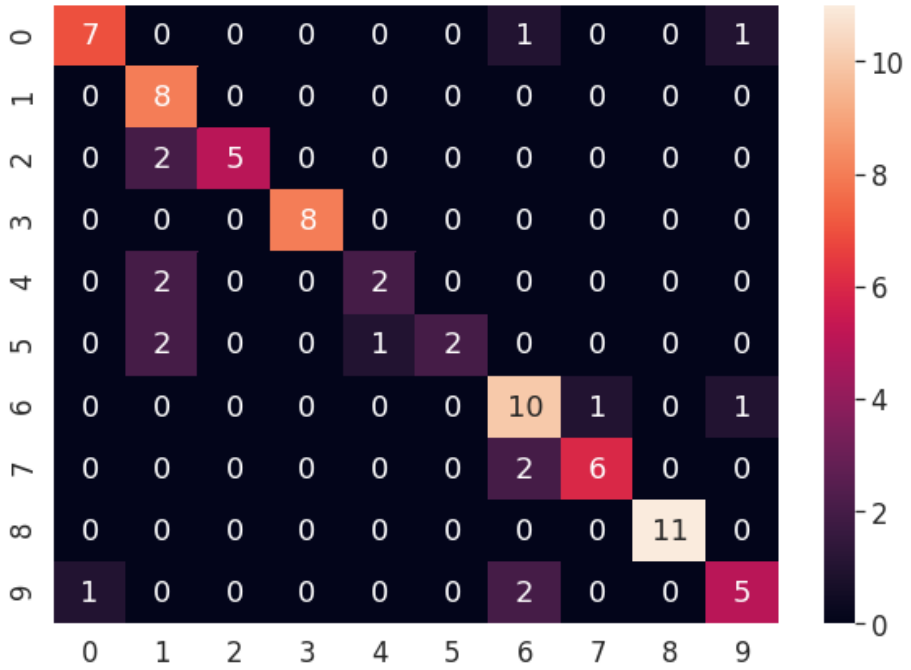

Figure 4: Romanian BERT confusion matrix.

Figure 5: Romanian BERT train and validation loss over 35 epochs.

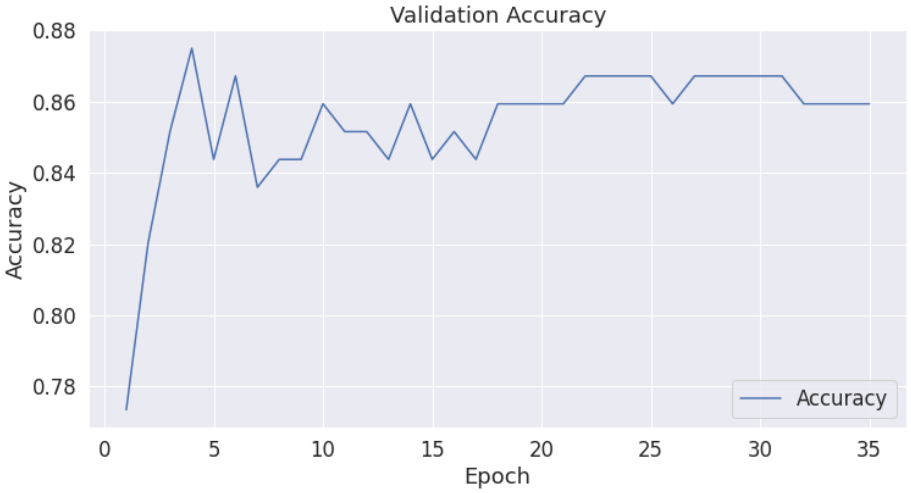

Figure 6: Romanian BERT train accuracy over 35 epochs.

3.2 English BERT

We got our English data by translating the original Romanian into English via Azure's Cognitive Services. The translated dataset is available here.

This time we used BERT base cased and the model got around 65% test accuracy. While signifincantly lower than both Romanian BERT and XLM-RoBERTa, we still considered it as an interesting result and decided to include it in the project report.

As it can be seen from the confusion matrix and loss and accuracy plots, the English BERT model is clearly underperforming compared to the other two BERT models.

Additionally, we used the same tokenizer and tokenizer parameters as for Romanian BERT.

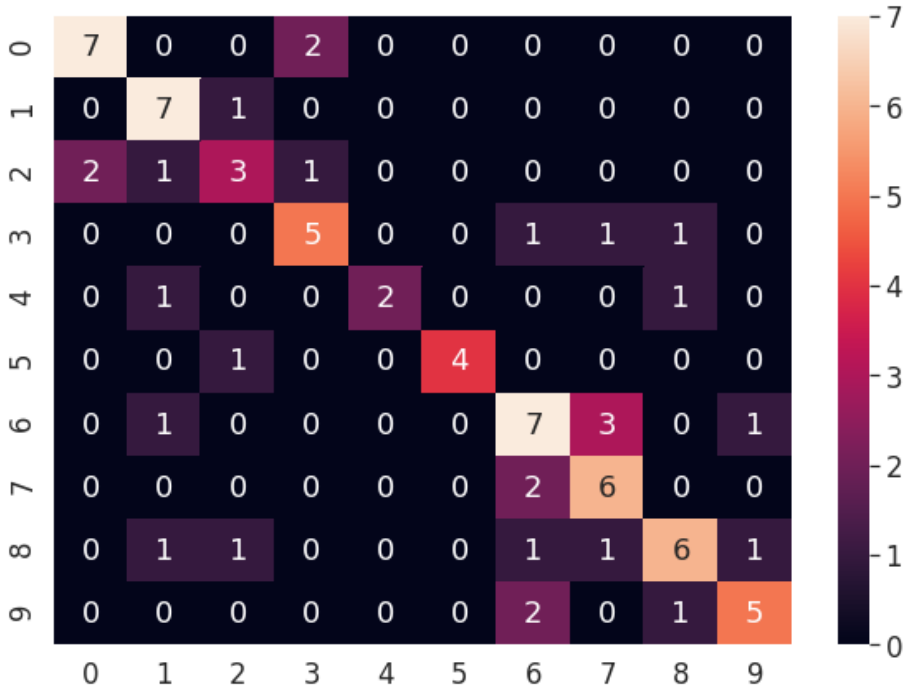

Figure 7: English BERT confusion matrix.

Figure 8: English BERT train and validation loss over 35 epochs.

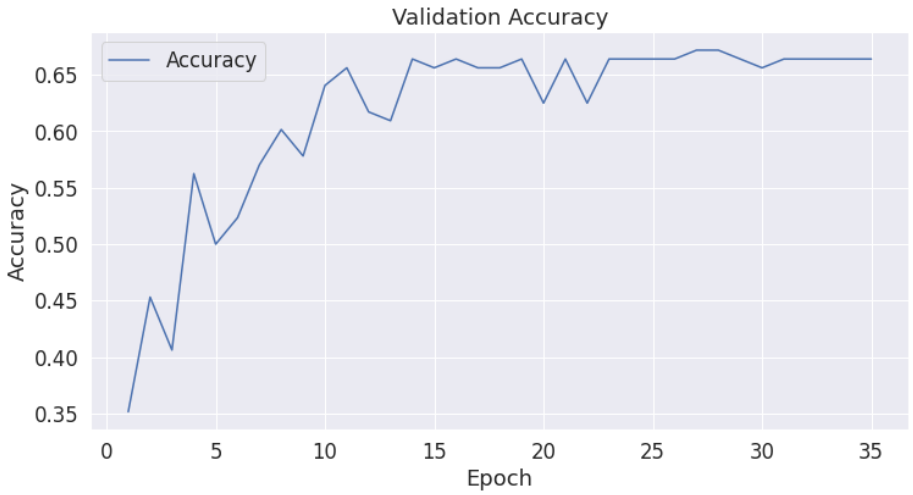

Figure 9: English BERT train accuracy over 35 epochs.

3.3 Scikit-learn MLP

Our initial idea was to use sci-kit learn's MLP as a second baseline for our network models, since it's also a deep neural network.

We used the same tf-idf representation as for the SGD model. We trained the MLP on raw data, so no preprocessing has been done whatsoever.

After observing that the MLP's results were substantially better than those of the XLM-RoBERTa ensemble, respectively an 84% accuracy compared to 83%, we decided to keep experimenting more with the model.

We increased both the number of the maximum iterations of the network from 200 to 500 and the number of hidden units from 100 to 200, but observed no improvement. We decided to use the 'invscaling' for the learning rate parameter, as per documentation. In the end we changed the random state parameter a few times until we achieved a maximum test accuracy of 88.75% and a test loss of 0.007.

After managing to obatin a comparable accuracy with that of XLM-RoBERTa model, we decided to use scikit-learn's MLP predictions as one of our submissions.

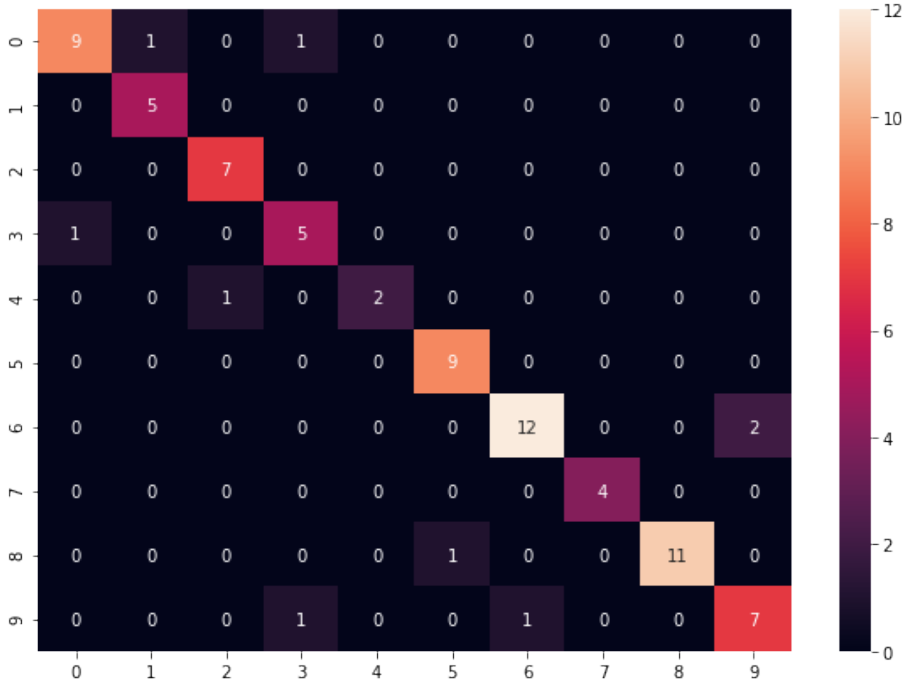

Figure 10: MLP confusion matrix.