UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Apostolos Koukouvinis*
sdi2000098

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester2023*

Contents

1. Abstract

The task is to create a sentiment classifier (NEUTRAL, NEGATIVE, POSITIVE) for greek tweets using BERT model. The two BERT models are given in the assignment : greek BERT and Greek-Distil-BERT. To tackle this we have to pre-process the data, create a Data Loader, encode the Sentiments using Label Encoder, create a wrapper for the BERT model to experiment with more hyperparameters, train the model and predict the test set Sentiments. To do that, we have dataset and validation set, based on these we will try to tune the hyperparameters using optuna framework to improve f1 score. We have chosen to improve f1 score since after experimenting a lot we found out that the most consistent score is f1. This is normal since it uses both precision and recall to calculate its value. After we find the best model and the best hyperparameters, we will concatenate the validation and dataset to train at the new dataset and predict the sentiments of the test set.

2. Data processing and analysis

2.1. Pre-processing

Basically the pre-process is simillar to the pre-process of the first, third and second assignment without using stemming , the results we got without stemmer are better, this is natural since BERT tokenizer has its own way to brake the words, so it better to be whole. First of all we lowercase all letters and remove stopwords (we have created a stopword list based on greek frequent words). Then we remove urls. We have to mention here that all the steps above were done for both data and validaiton set, and of course test set.

2.2. Analysis

Since the dataset is the same of the first assignment, the analysis is also the same. After analysis on the words frequency, we could not find any relationship between the 20 most frequent words and any sentiment, so we can drop these words or just dont mind them. It is also interesting, that after the data cleaning, the top 3 most frequent words per sentiment are common. So the dataset is prety balanced in terms of word frequency

2.3. Data partitioning for train, test and validation

We are keeping the original partition in train and validation set so the optuna studies are more simple.

2.4. Vectorization

In terms of BERT model, vectorization refers to the way we are tokenizing and thene ncoding the text. So the process is this : The process begins with the preparation of text data using the greek BERT tokenizer by transforming raw texts into a format that BERT can process. Initially, each piece of text is tokenized, converting it into a sequence of tokens that represent words or subwords, based on the vocabulary that the BERT model

has been trained on. Following tokenization, the sequences are analyzed to identify the longest sequence among them, which determines the uniform length to which all token sequences will be padded. This ensures consistency, as BERT requires input sequences of the same length. Subsequently, each text is encoded into a series of numeric IDs that correspond to its tokens, alongside an attention mask. The attention mask indicates to the model which tokens are actual data and which are padding added to achieve uniform sequence length. The above process is reapeated but tokenizer for DISTIl BERT is used instead of the simple BERT tokenizer. After this process we need to define loaders for dataset and validation set. The process is this : create sentiment labels from training and validation datasets into tensors, also convert tokens and attention mask to tensors. Then define properly the dataloaders using RandomSampler for dataloader (for random batches) and SequentialSampler for validation loader (for sequential batches).

3. Algorithms and Experiments

The first experiment is pretty basic, it is based on the Chris McCormick's tutorial (referenced at a piazza post), which uses the pretrained model. The results we got using this model, are pretty descent, in fact they are better than more complex models. We use optuna to tune some hyperparametes, for this simple model the only hyperparameters are learning rate and batch size. We are using the Adam optimizer, and we will experiment with more optimizers later. The result of the optuna study is not present in the notebook that is submitted. This happens bcause each optuna study takes about 10 hours, meaning that kaggle would fail to run the whole notebook. So for this project, all optuna studies are commented out. However I am going to present in the report all the results and some comments on them. Keep in mind that the models are trained for 2 epochs in each optuna study. The inital thought was with 5 epochs, but the results were not consistent. For same hyperparameters, the first 10 trials of the optuna study with 5 epochs for BERT model yielded these results :

```
[I 2024-03-05 17:25:08,019] Trial 0 finished with value: 0.34381671554562143 and parameter
s: {'learning_rate': 1.0188948283567978e-06, 'batch_size': 32, 'hidden_size': 256, 'dropou
t_rate': 0.2154438241818264, 'activation_function': 'leaky_relu'}. Best is trial 0 with va
lue: 0.34381671554562143.
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: Undefined
MetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
[I 2024-03-05 17:40:58,027] Trial 1 finished with value: 0.16666666666666666 and parameter
s: {'learning_rate': 6.3593251150694e-05, 'batch_size': 16, 'hidden_size': 512, 'dropout_r
ate': 0.4868772878950397, 'activation_function': 'gelu'}. Best is trial 0 with value: 0.34
381671554562143.
[I 2024-03-05 17:56:49,506] Trial 2 finished with value: 0.337110974207039 and parameters:
{'learning_rate': 6.95052966838839e-06, 'batch_size': 16, 'hidden_size': 512, 'dropout_rat
e': 0.2744281308590314, 'activation_function': 'tanh'}. Best is trial 0 with value: 0.3438
1671554562143.
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: Undefined
MetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
[I 2024-03-05 18:12:38,428] Trial 3 finished with value: 0.16666666666666666 and parameter
s: {'learning_rate': 7.312633679939705e-05, 'batch_size': 16, 'hidden_size': 512, 'dropout
_rate': 0.3228439471023018, 'activation_function': 'relu'}. Best is trial 0 with value: 0.
34381671554562143.
[I 2024-03-05 18:27:39,943] Trial 4 finished with value: 0.39185003511799005 and parameter
s: {'learning_rate': 1.5368069610276913e-05, 'batch_size': 32, 'hidden_size': 256, 'dropou
t_rate': 0.15876517276227778, 'activation_function': 'gelu'}. Best is trial 4 with value:
0.39185003511799005.
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1344: Undefined
MetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted s
amples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
[I 2024-03-05 18:42:48,262] Trial 5 finished with value: 0.2925447725949313 and parameter
s: {'learning_rate': 8.861542419389354e-05, 'batch_size': 64, 'hidden_size': 512, 'dropout
_rate': 0.3546595307589605, 'activation_function': 'relu'}. Best is trial 4 with value: 0.
39185003511799005.
[I 2024-03-05 18:58:39,573] Trial 6 finished with value: 0.32489108960535806 and parameter
s: {'learning_rate': 3.303676210531466e-05, 'batch_size': 16, 'hidden_size': 512, 'dropout
_rate': 0.19848922565268334, 'activation_function': 'relu'}. Best is trial 4 with value:
0.39185003511799005.
[I 2024-03-05 19:13:41,051] Trial 7 finished with value: 0.4018340626563698 and parameter
s: {'learning_rate': 2.3314095101024966e-05, 'batch_size': 32, 'hidden_size': 768, 'dropou
t_rate': 0.17885568914471325, 'activation_function': 'tanh'}. Best is trial 7 with value:
0.4018340626563698.
[I 2024-03-05 19:29:32,313] Trial 8 finished with value: 0.304736348618673 and parameters:
{'learning_rate': 3.0111919943709186e-05, 'batch_size': 16, 'hidden_size': 256, 'dropout_r
ate': 0.4344260226186609, 'activation_function': 'relu'}. Best is trial 7 with value: 0.40
18340626563698.
[I 2024-03-05 19:45:24,090] Trial 9 finished with value: 0.39307247233003767 and parameter
s: {'learning_rate': 1.4594813670136401e-05, 'batch_size': 16, 'hidden_size': 256, 'dropou
t_rate': 0.3528756042599036, 'activation_function': 'tanh'}. Best is trial 7 with value:
```

As we see there are trials with really low resutls, like 0.16666. This may happen because overfitting occurs. While for 2 epochs the results are :

```
[I 2024-03-11 21:02:47,496] Trial 0 finished with value: 0.3857802344157447 and parameter
s: {'learning_rate': 9.323448145055059e-06, 'batch_size': 24, 'hidden_size': 640, 'dropout
_rate': 0.15315819482768925, 'activation_function': 'relu'}. Best is trial 0 with value:
0.3857802344157447.
[I 2024-03-11 21:18:22,304] Trial 1 finished with value: 0.374537981119021 and parameters:
{'learning_rate': 8.288240929997511e-06, 'batch_size': 16, 'hidden_size': 640, 'dropout_ra
te': 0.1911920557212903, 'activation_function': 'relu'}. Best is trial 0 with value: 0.385
7802344157447.
[I 2024-03-11 21:33:30,449] Trial 2 finished with value: 0.37306838933637904 and parameter
s: {'learning_rate': 8.599610686602627e-06, 'batch_size': 24, 'hidden_size': 640, 'dropout
_rate': 0.16345728429719142, 'activation_function': 'relu'}. Best is trial 0 with value:
0.3857802344157447.
[I 2024-03-11 21:48:38,449] Trial 3 finished with value: 0.39131903235207816 and parameter
s: {'learning_rate': 1.0527609325529819e-05, 'batch_size': 24, 'hidden_size': 640, 'dropou
t_rate': 0.17055257890829742, 'activation_function': 'relu'}. Best is trial 3 with value:
0.39131903235207816.
[I 2024-03-11 22:03:46,036] Trial 4 finished with value: 0.39586876611426514 and parameter
s: {'learning_rate': 1.2536688824976823e-05, 'batch_size': 24, 'hidden_size': 640, 'dropou
t_rate': 0.17095496548626976, 'activation_function': 'relu'}. Best is trial 4 with value:
0.39586876611426514.
[I 2024-03-11 22:19:20,978] Trial 5 finished with value: 0.3386723277661187 and parameter
s: {'learning_rate': 1.4391818708563502e-05, 'batch_size': 16, 'hidden_size': 640, 'dropou
t_rate': 0.17694952175908585, 'activation_function': 'relu'}. Best is trial 4 with value:
0.39586876611426514.
[I 2024-03-11 22:34:29,283] Trial 6 finished with value: 0.36325692297604445 and parameter
s: {'learning_rate': 1.1917133730052962e-05, 'batch_size': 24, 'hidden_size': 512, 'dropou
t_rate': 0.18852524811271903, 'activation_function': 'relu'}. Best is trial 4 with value:
0.39586876611426514.
[I 2024-03-11 22:49:13,854] Trial 7 finished with value: 0.36196651183639095 and parameter
s: {'learning_rate': 8.970443991351446e-06, 'batch_size': 32, 'hidden_size': 512, 'dropout
_rate': 0.19292565703003986, 'activation_function': 'tanh'}. Best is trial 4 with value:
0.39586876611426514.
[I 2024-03-11 23:04:21,950] Trial 8 finished with value: 0.3982512154815223 and parameter
s: {'learning_rate': 1.4593761856557857e-05, 'batch_size': 24, 'hidden_size': 512, 'dropou
t_rate': 0.15785011576269473, 'activation_function': 'relu'}. Best is trial 8 with value:
0.3982512154815223.
[I 2024-03-11 23:19:57,143] Trial 9 finished with value: 0.3716734279417408 and parameter
s: {'learning_rate': 1.1097227464658155e-05, 'batch_size': 16, 'hidden_size': 512, 'dropou
t_rate': 0.19964534281289661, 'activation_function': 'relu'}. Best is trial 8 with value:
0.3982512154815223.
```

As we see the results are consistently higher than 0.33, with most of them being around 0.37. This is not weird, since it's relatively common for models like BERT to overfit if they are trained for too many epochs on a dataset. BERT models, due to their large number of parameters and deep understanding of language from pre-training, can quickly learn and even memorize details from a specific, possibly small training dataset. When fine-tuning on such datasets, just a few epochs can lead to overfitting, as the model starts fitting to the noise or specific idiosyncrasies of the training data rather than learning generalizable patterns. Furthermore, the last optuna study of the project examines exactly this, it tries to optimize learnign rate and epochs, after several re runs the best results were with 2 or 3 epochs. Anyways we continue with 2 epochs.

So for the first optuna study for this simple model we got these results :

```
Best trial:
  F1 Score:  0.40504371244045967
  Precision:  0.4141407487028667
  Recall:  0.41093272171253825
  Accuracy:  0.41093272171253825
  Params:
    learning_rate: 2.0786351576940027e-06
    batch_size: 64
```

Next, we will use a wrapper for the model to add more features this is the class BertClassifier : At its core, the model employs the pre-trained BERT layer for extracting complex language patterns, augmented with a dropout layer to prevent overfitting. A linear transformation adjusts the dimensionality to a predefined hidden size, followed by a choice of activation functions (ReLU, GELU, Tanh, or LeakyReLU) to introduce nonlinearity. The architecture culminates in a classifier layer that maps the processed features to sentiment categories. The model implements just on hidden layer, trials with more hidden layers turned out fruitless, we faced this issue in the second assignment as well, in which more hidden layers yielded worse results. In fact, in this exercise, for this dataset, the naive model we used in the first experiment, might yield the best results. Anyways, for this more complex BERT wrapper we got these results :

```
Best trial:
  F1 Score:  0.4003667473297847
  Precision:  0.4066172730115328
  Recall:  0.40386085626911317
  Accuracy:  0.40386085626911317
  Params:
    learning_rate: 1.2045524369398105e-05
    batch_size: 16
    hidden_size: 512
    dropout_rate: 0.1799444890762208
    activation_function: gelu
```

Keep in mind that so far we are not experimentig at all with the optimizers, in fact we are not going to experiment with them at all, since one optuna study we tried with different optimizers, yielded terrible results, so Adam is perhaps the best choice. Plus, in the third assignment, we were asked to use just this. For the third experiment we are using essentially the same classifier, configured for distil Bert models. The result is this :

```
Best trial:
  F1 Score:  0.3802508075520941
  Precision:  0.38772008574330485
  Recall:  0.3849388379204893
  Accuracy:  0.3849388379204893
  Params:
    learning_rate: 1.525057541628784e-05
    batch_size: 64
    hidden_size: 512
    dropout_rate: 0.49831817398331124
    activation_function: tanh
```

Could have been better... So we try to narrow down the value pools of the hyperparameters based on the best results we got from the optuna stuudy. This will yield this best trial :

```
Best trial:
  F1 Score:  0.37701045901798047
  Precision:  0.37805410462558664
  Recall:  0.37824923547400613
  Accuracy:  0.37824923547400613
  Params:
    learning_rate: 1.3782593313403278e-05
    batch_size: 128
    hidden_size: 400
    dropout_rate: 0.5210426075116331
    activation_function: gelu
```

Even worse... I have no clear explanation how this may happen, however I believe is due to the misleading dataset. Finally, after many trials with the GREEK-Bert and different hyperaparameters, we came up with this best trial :

```
[I 2024-03-05 21:47:36,622] Trial 17 finished with value: 0.40933991050641144 and paramete
rs: {'learning_rate': 1.7526138202840862e-05, 'batch_size': 16, 'hidden_size': 256, 'dropo
ut_rate': 0.17450016248069433, 'activation_function': 'tanh'}. Best is trial 17 with valu
e: 0.40933991050641144.
```

3.0.1. Table of trials. Before presenting the table of trials, we need to mention that we are not going to use learning scheduler or earlys topping, adding thses features in this and previous assignments, yielded worse results. The Scores presented to the table of trial are f1 scores, and it is the mean value of 2-3 different re-runs of the optuna study :

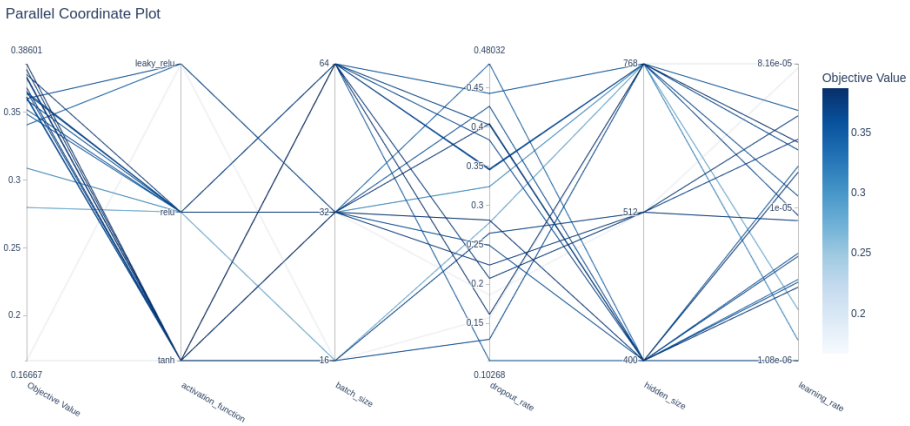
Trial	Description	Score
1	Naive Greek BERT	0.402
2	Greek BERT with wrapper	0.406
3	Distil GREEK bert with wrapper	0.389

Table 1: Trials

3.1. Hyper-parameter tuning

For hyperparameter tuning, I used the optuna framework to tune learning rate, batch size, hidden size, dropout rate and activation function chosen. Some intersting plots provided by optuna are these :

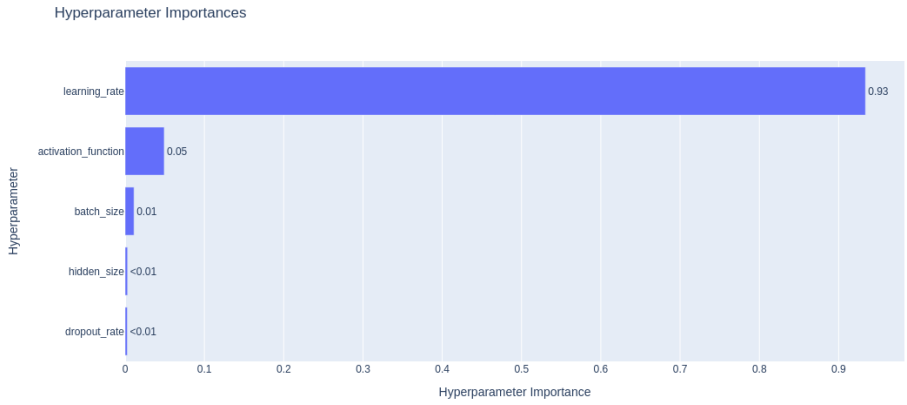
Parallel coordinate plot



The parallel coordinate plot visualizes multi-dimensional data points, such as the results of hyperparameter tuning, by placing each hyperparameter on its own axis and drawing lines across the axes for each combination of hyperparameter values tested. From the plot above we can make some assumptions : the learning rate values are concentrated towards the lower end of the scale, which suggests that smaller learning rates may be contributing to higher F1 scores. There is a mix of batch sizes throughout the trials. It doesn't show a clear trend towards which batch size is better based on the F1 score, suggesting that batch size may not be the most significant factor in the model's performance. The trials with higher F1 scores appear more frequently with certain hidden sizes. It looks like hidden sizes of 768 and 512 are more common among the

better-performing trials, indicating these might be more effective sizes for the hidden layer in this classification task. Lower dropout rates seem to be associated with higher F1 scores, with rates closer to 0.1 appearing more frequently among better-performing trials. This could imply that too much regularization via dropout may not be beneficial for this task. It seems that relu and leaky relu are the activation functions more frequently associated with the higher F1 scores, indicating that these functions might be better suited for this model and task. You can see a range of objective values, but the trend seems to be that combinations of lower learning rates, moderate hidden sizes, and lower dropout rates with relu or leaky relu activation functions are associated with higher F1 scores.

Hyperparameter importance polot



The Hyperparameter Importances bar chart provides a clear visualization of the relative importance of each hyperparameter in the optimization process. Here, we see that the learning rate is by far the most influential hyperparameter with an importance score close to 0.93, indicating a strong impact on the model's performance. This suggests that choosing the right learning rate is critical for the success of the model in this particular sentiment analysis task. The other hyperparameters — activation function, batch size, hidden size, and dropout rate — have significantly lower importance scores, all below 0.05. This implies that, in the context of this specific optimization study, variations in these parameters have a comparatively negligible effect on the model's F1 score.

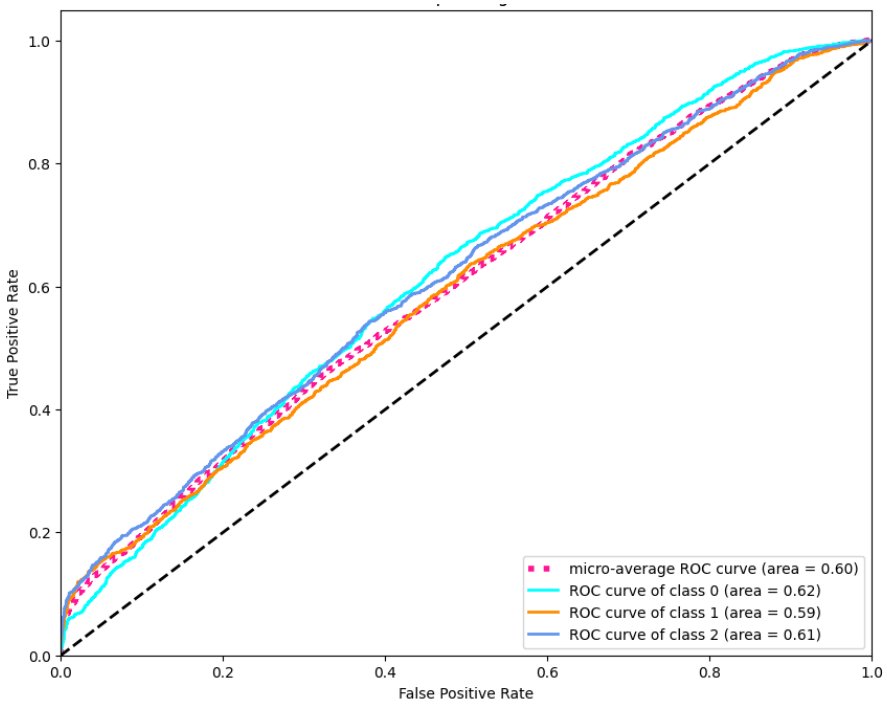
3.2. Optimization techniques

Mainly optuna.

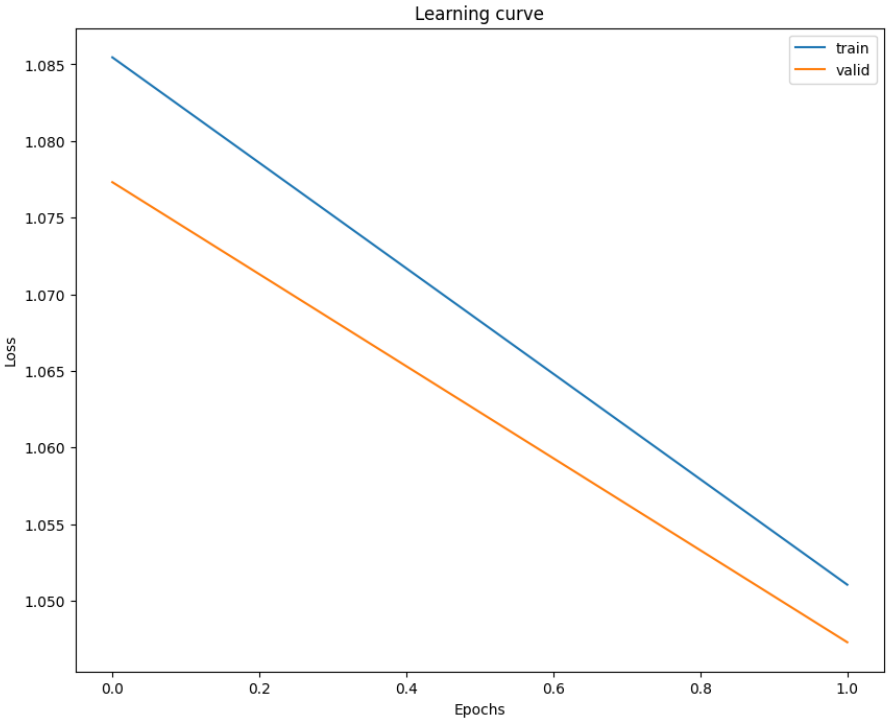
3.3. Evaluation

For evaluation I used f-score and the scores are close to 0.40 without exceding 0.41. There is no overftting or underffiting in the model and this can be proven from he diagrams and curves. Keep in mind, that adding a lot of complexity in my model, like regularization, learning rate scheduler and stuff like thatm would make the GPU run out of memory. However, after experimenting a lot withi learnign rate schedulers and

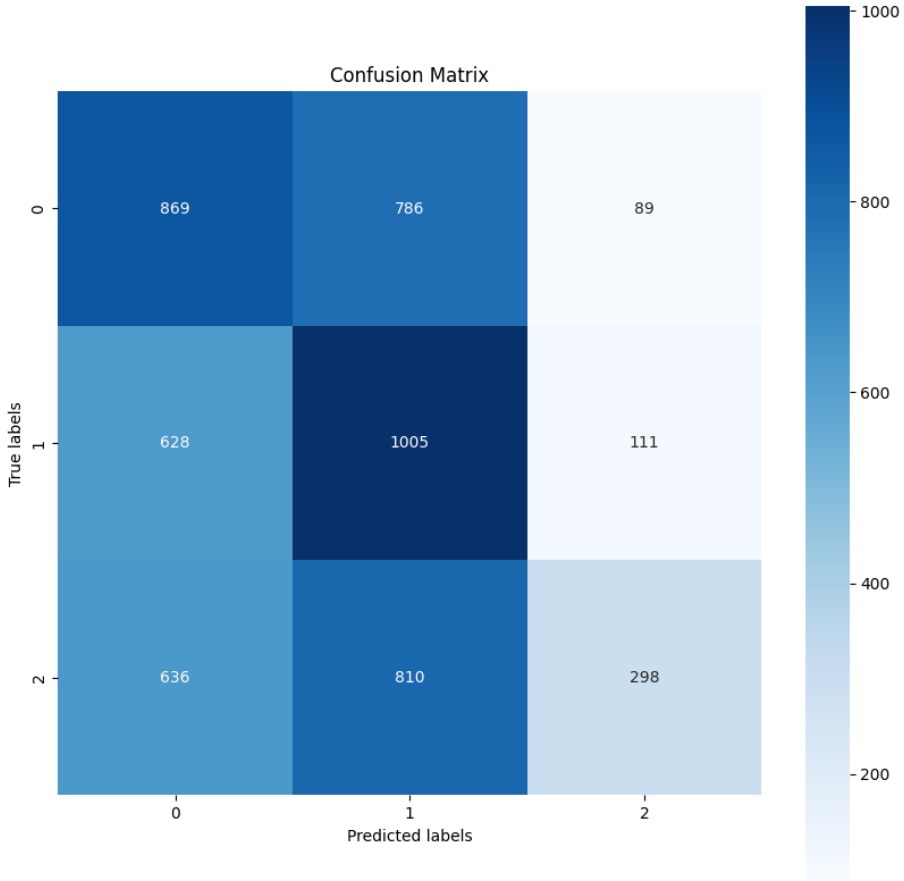
early stopping, it turned out that a good number of epochs, so the model doesnt underfit or overfit, is epochs = 2. I will be using this. It is interesting, that when number of epochs is increased, the results from optuna studies are not consistent. Some may be up to 0.4 while other just drom at 0.16. 2 epochs is a good number that makes training fast while results remain good.



3.3.1. ROC curve. The curves for three different classes are shown, with the areas under the curve (AUC) for class 0, class 1, and class 2 being 0.62, 0.59, and 0.61, respectively. These AUC values are moderately above 0.5, which represents the performance of a random classifier, indicating that the classifier performs better than random chance, but not by a wide margin. The micro-average ROC curve, with an AUC of 0.60, suggests overall performance that is similarly modest. Given that the curves are relatively close to the diagonal line of no-discrimination, it suggests there's room for improvement in the model's ability to discriminate between classes. Of course, for this dataset these results were expected.



3.3.2. Learning Curve. The learning curve displayed shows a consistent decrease in loss for both training and validation datasets over epochs, suggesting that the model is learning and generalizing well so far. Notably, there is no apparent divergence between the training and validation loss, which indicates that the model is not overfitting — typically characterized by a significant gap where the validation loss increases or plateaus while the training loss continues to decrease. Since both curves are nearly parallel and the gap between them is minimal, the model also does not seem to be underfitting, which would be indicated by high training and validation loss.



3.3.3. Confusion matrix. The confusion matrix displays the performance of the model across three classes (0, 1, and 2). Given that 0 corresponds to 'NEGATIVE', 1 to 'NEUTRAL', and 2 to 'POSITIVE', the confusion matrix shows that the classification model performs best in correctly identifying 'NEUTRAL' sentiments, with 1005 instances correctly classified. However, there is notable confusion between 'NEGATIVE' and 'NEUTRAL' categories, with 786 'NEGATIVE' instances being misclassified as 'NEUTRAL'. Additionally, 'POSITIVE' sentiment is also frequently confused with 'NEUTRAL', with 810 instances mislabeled. This could indicate that the features distinguishing 'NEGATIVE' and 'POSITIVE' sentiments from 'NEUTRAL' are not being effectively captured by the model, or it could reflect an inherent difficulty in distinguishing between these sentiment classes, possibly due to overlapping linguistic expressions in the dataset used for training.

4. Results and Overall Analysis

4.1. Results Analysis

The results remain poor, same with the first 3 projects. A f-score close to 0.4 is just slightly better than random guessing. This mainly happens because of the mislabeled dataset.

4.1.1. Best trial. The best trials are snapshoted and presented in the Algorithms and Experiments section. Generally, the BERT model yields better results while the DISTIL version of it is pretty faster in training. The decision between the two models relies mainly on the time constraints we have.

4.2. Comparison with the first project

The results of the BERT model seem to be better than those of the Logistic Regression model. This finding aligns well with theoretical expectations, given BERT's advanced deep learning framework designed specifically for understanding the intricacies of human language. Unlike logistic regression, which utilizes a straightforward linear combination of input features, BERT incorporates a transformer architecture that enables it to analyze words in relation to the entirety of the text. This bidirectional understanding of context allows for a nuanced interpretation of language nuances, significantly enhancing its performance on complex NLP tasks. Moreover, BERT's pretraining on extensive text corpora equips it with a broad linguistic foundation, which, when fine-tuned, can be adeptly applied to specific project requirements.

4.3. Comparison with the second project

The results of the BERT model seem to be better than those of the deep Neural Netwok model. BERT's superiority can be attributed to its advanced architecture and pretraining approach, which are specifically tailored for understanding the complexities of language. While deep neural networks can capture nonlinear relationships and feature interactions through multiple layers of processing, they do not inherently account for the contextual nature of language. BERT, however, leverages the transformer architecture, which employs attention mechanisms to analyze words in relation to all other words in a sentence, thereby capturing nuances, ambiguities, and the context of language use more effectively. Additionally, BERT is pretrained on a large corpus of text data using tasks designed to understand language context and relationships, such as predicting missing words in a sentence. This pretraining imbues BERT with a rich understanding of language before it is even fine-tuned for specific tasks.

4.4. Comparison with the third project

The results of the BERT model seem to be better than those of the bidirectional stacked Recurrent Neural Network (RNN). This happens mainly due to BERT's advanced architectural features and its pretraining approach, both of which are intricately designed to capture the complexities and contextual nuances of language more effectively than RNN-based models, even those enhanced with bidirectional processing and stacking strategies. LSTM and GRU process text sequentially, which allows them to capture information from both past and future contexts. However, their sequential nature can lead to limitations in capturing long-range dependencies within text, due to issues like vanishing gradients, where the model's ability to learn from data that is far apart in the input sequence is diminished. BERT, on the other hand, leverages the Transformer architecture, which employs self-attention mechanisms. This design enables BERT to consider all words in a sentence simultaneously, making it inherently better at understanding the full context of each word, regardless of their position or distance

from each other in the text. This parallel processing capability not only improves the model's efficiency but also its effectiveness in capturing complex linguistic patterns and dependencies.