UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Ioanna Poulou*
sdi: sdi2100161

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

<Briefly describe what's the task and how you will tackle it.>
The goal of the task is to create a sentiment classifier by fine-tuning two models , **the GreekBERT** and the **DistilGREEK-BERT**, for a twitter dataset about the Greek general elections. In the first assignment, we approached the same subject using logistic regression, in the second we used deep neural networks and in the third we utilized a stacked RNNs. However, in this Project, we are going to fine-tune the above models, in order to examine if they would lead to better and more efficient results than these of the previous assignments. It is worth mentioning that both models belong in the category of **Bert** models. Bert is a natural language processing model, which considers the context of words in a bidirectional manner and it is mainly used for text classification, named entity recognition and question answering tasks.

2. Data processing and analysis

2.1. Pre-processing

<In this step, you should describe and comment on the methods that you used for data cleaning and pre-processing. In ML and AI applications, this is the initial and really important step.

For example some data cleaning techniques are: Dropping small sentences; Remove links; Remove list symbols and other uni-codes.>
Before training my model, I cleaned my data, in order to increase the quality of them. In the previous Projects, I executed an exhaustive pre-processing phase in order to improve the dataset. However, in this Project the data cleaning is much more simpler, since the model learns from the context provided by surrounding words. Therefore, even stop words or small sentences are necessary for the model in order to understand the context of each tweet. More specifically, I selected to remove from the data links, mentions and emojis, because they do not facilitate the understanding of the tweet. Moreover, I employed the **encode plus** function, which not only tokenizes the input text into subwords or words but also incorporates the tokens **[CLS]** and **[SEP]** at the begin and the end of each tweet . Furthermore, it handles padding and uses an attention mask in order to let the model know which tokens represent real words in the sequence.

2.2. Analysis

<In this step, you should also try to visualize the data and their statistics (e.g., word clouds, tokens frequency, etc). So the processing should be done in parallel with an analysis. >
It is worth noting that before and after Pre-processing, I visualized the data using **word clouds** and **tokens frequency diagrams**.In this way, i could understand how efficient the data clean was and it helped me to make Pre-processing better. In essence, word clouds provided me with the ability to discern which elements of the sentences were important for Bert model and which should be removed. The word clouds before and

after the data cleaning are listed below, while you can create tokens frequency diagrams by running the related code in my notebook.

Figure 1: Word Cloud before Preprocessing

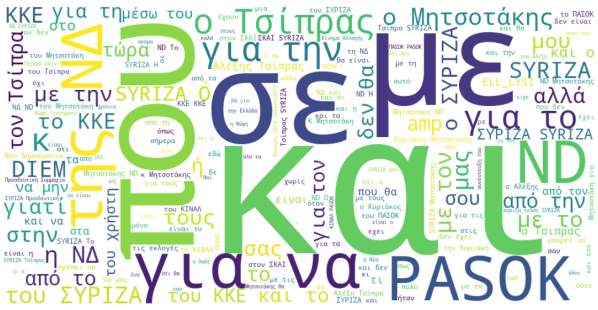


Figure 2: Word Cloud after Preprocessing

2.3. Data partitioning for train, test and validation

<Describe how you partitioned the dataset and why you selected these ratios>
I let the train and the validation datasets as they was given to us. The ratio **80:20** is the optimal, since we want variety of training data, but we also don't want our model overfits them. It is worth noting that I concatenated the columns "Party" and "Text", in order to make the the procedure of Vectorization easier.

2.4. Vectorization

<Explain the technique used for vectorization>
Vectorization is the technique, which converts text data into numerical representations. In our model, the **encoder plus** function has the role of the vectorizer, because in the attempt to convert the data into a suitable format, the function ultimately yields a numerical representation of the input.

3. Algorithms and Experiments

3.1. Experiments

<Describe how you faced this problem. For example, you can start by describing a

first brute-force run and afterwords showcase techniques that you experimented with. **Caution:** we want/need to see your experiments here either they increased or decreased scores. At the same time you should comment and try to explain why an experiment failed or succeeded. You can also provide plots (e.g., ROC curves, Learning-curves, Confusion matrices, etc) showing the results of your experiment. Some techniques you can try for experiments are cross-validation, data regularization, dimension reduction, batch/partition size configuration, data pre-processing from 2.1, gradient descent>

In this section, i will represent you the results of some techniques i used for both models in order to find out the best combination. It is worth noting that for speed reasons all the experiments, excluding the final ones, are not included in the notebook. Moreover, I planned to conduct experiments using the GreekBERT model firstly, followed by an exploration with the DistilGREEK-BERT model.

3.1.1. GreekBERT model.

- **Batch size:** The batch size refers to the number of input samples processed in a single iteration during training. For fine-tuning BERT on a specific task, the authors recommend a batch size of 16 or 32, so I tried both of them. As it was expected, the batch size of 16 required more time than the batch size of 32. Surprisingly, both scenarios yielded almost identical outcomes, but the batch size of 32 exhibited a little smaller degree of overfitting, so I decided that the best choice is to set the batch size equal to 32. Here are the result for both batch sizes with learning rate 2e-5:

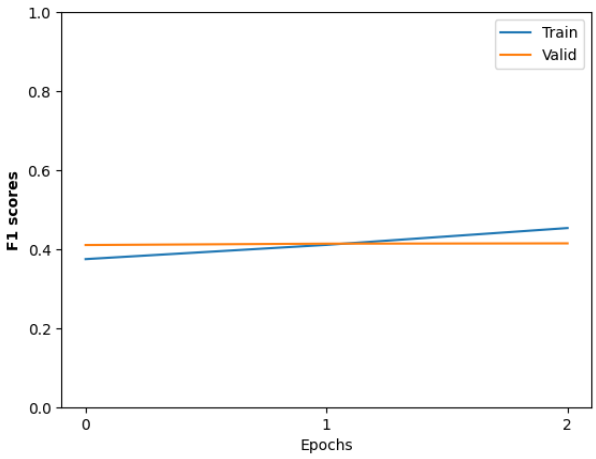


Figure 3: Batch Size 32: F1 Learning Curves

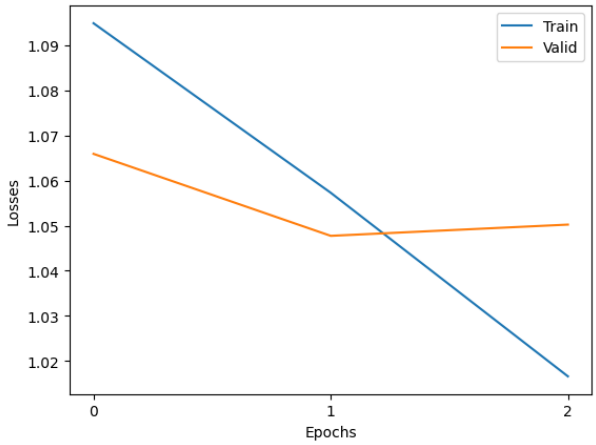


Figure 4: Batch Size 32: Loss Learning Curves

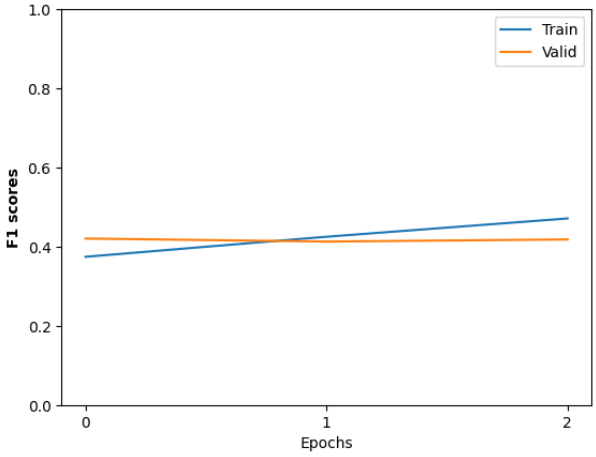


Figure 5: Batch Size 16: F1 Learning Curves

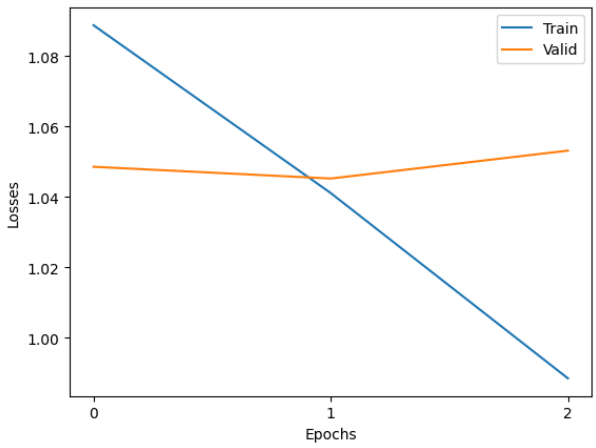


Figure 6: Batch Size 16: Loss Learning Curves

- **Learning Rate:** The learning rate is a hyperparameter that regulates the step size at each iteration while moving toward a minimum of a loss function. For the Adam optimizer the recommended learning rates are 2e-5, 3e-5, and 5e-5. The

difference observed among the results with the three different learning rates was not colossal. However, examining the details, I noticed that the f1 score reached 0.41 for the learning rate 2e-5, whereas the 3e-5 and 5e-5 learning rates achieved a little higher score of 0.42. Moreover, the 3e-5 learning rate demonstrated less overfitting than the learning rate of 5e-5, evidenced in both learning curves and for the these reasons, I concluded that the model corresponds better when the learning rate is equal to 3e-5.

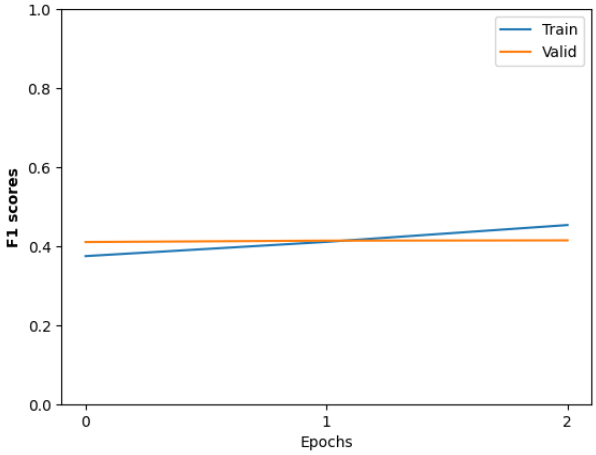


Figure 7: Learning rate 2e-5: F1 Learning Curves

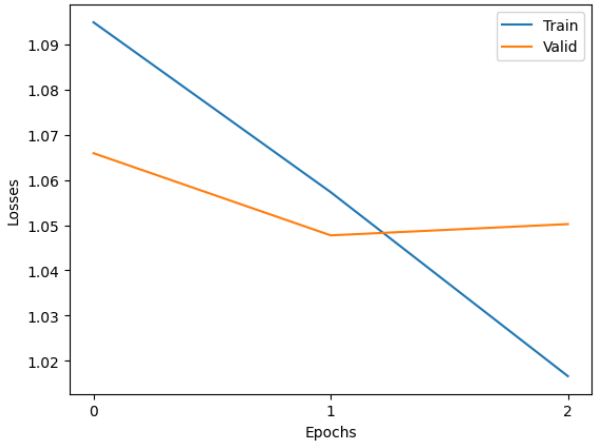


Figure 8: Learning rate 2e-5: Loss Learning Curves

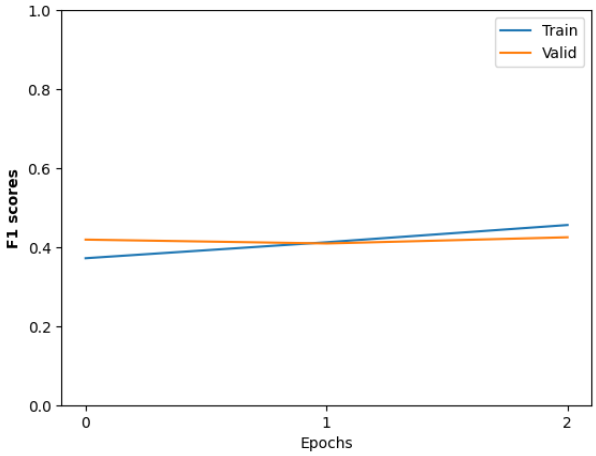


Figure 9: Learning rate 3e-5: F1 Learning Curves

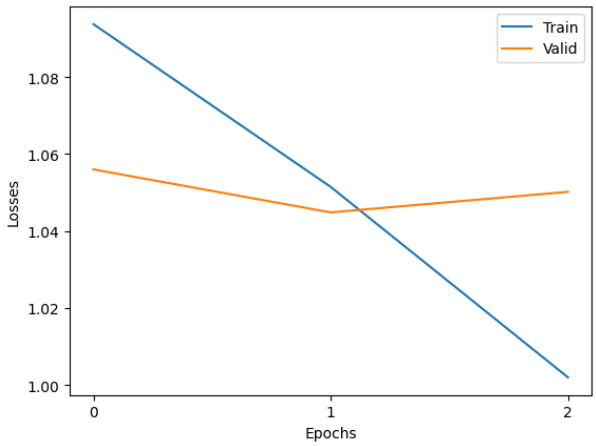


Figure 10: Learning rate 3e-5: Loss Learning Curves

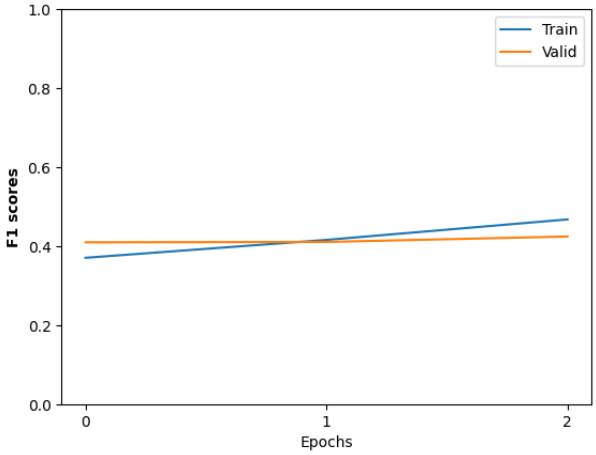


Figure 11: Learning rate 5e-5: F1 Learning Curves

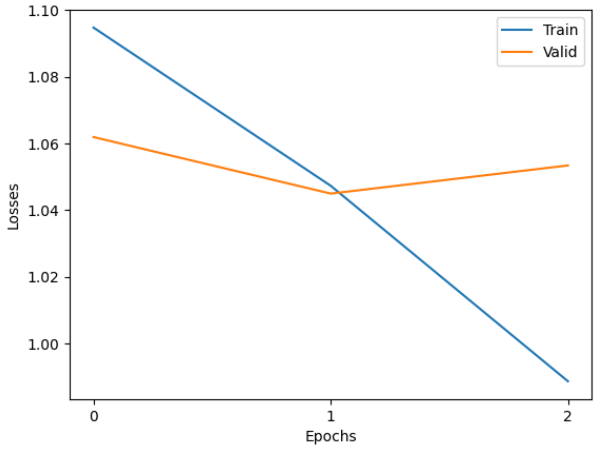


Figure 12: Learning rate 5e-5: Loss Learning Curves

- **Padding and Truncation:** Padding and Truncation are both methods that are used when working with variable size data. Padding adds special tokens to the sequences with the aim of all having the same length, while truncation removes tokens from the sequences that outstrip a predefined maximum length. In our scenario, where the maximum sentence length is 140 tokens, I conducted experiments with various parameter values, such as 60, 100, and 140 and I found out that with minor differences a maximum length of 60 created the best results. However, this implies that the model would neglect two-thirds of the tokens, which is generally considered suboptimal, so I decided that the best choice is to set the parameter **max length** equal to 140. In other words, the selected approach avoids to truncate any sentence.

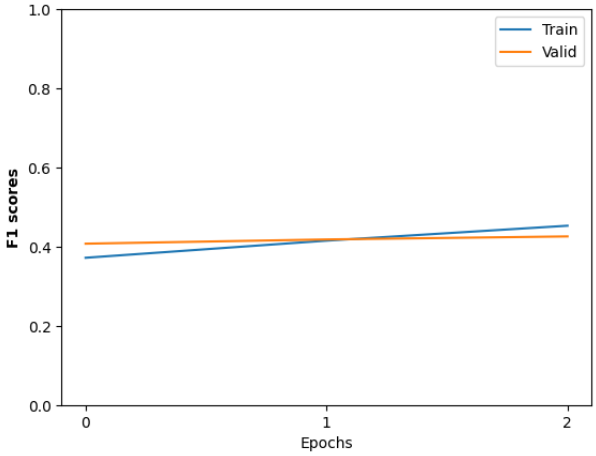


Figure 13: Max Length 60: F1 Learning Curves

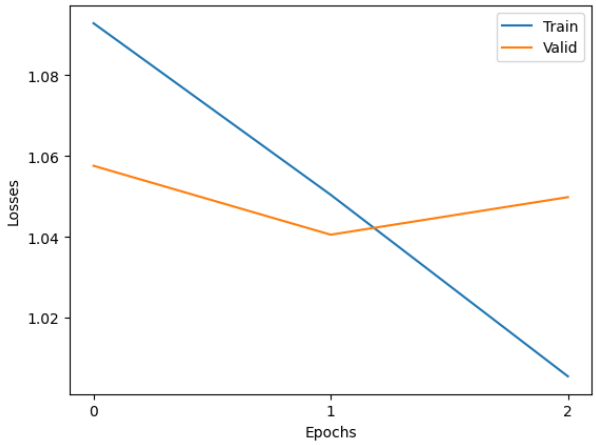


Figure 14: Max Length 60: Loss Learning Curves

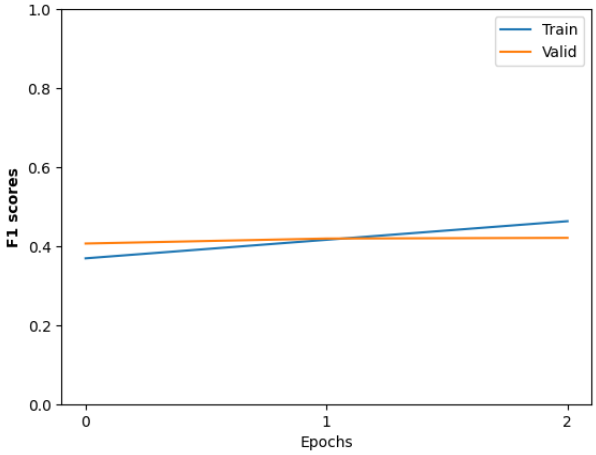


Figure 15: Max Length 100: F1 Learning Curves

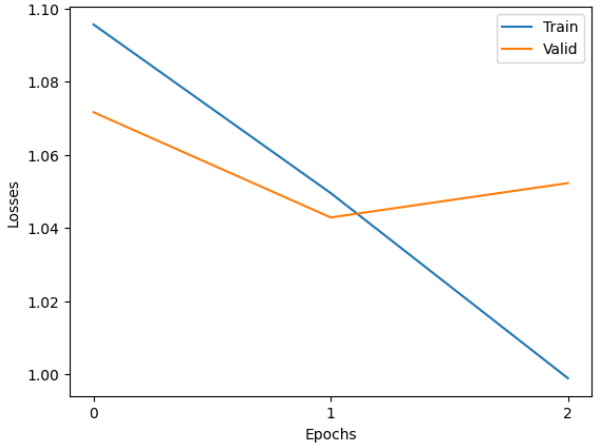


Figure 16: Max Length 100: Loss Learning Curves

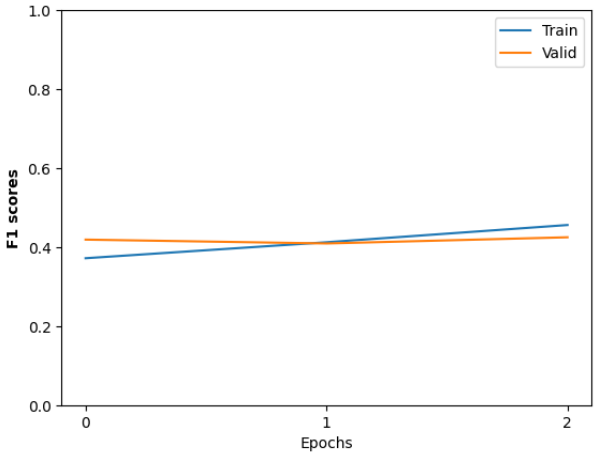


Figure 17: Max Length 140: F1 Learning Curves

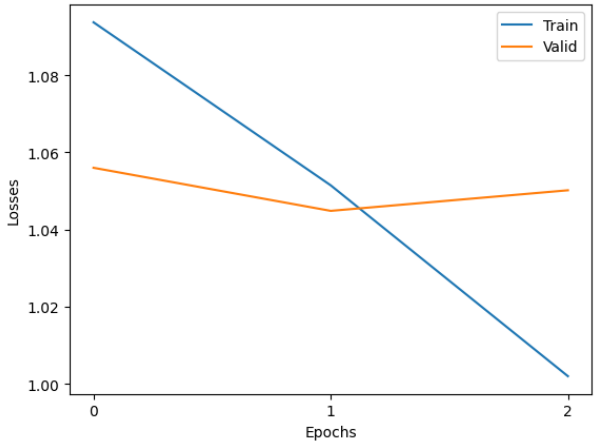


Figure 18: Max Length 140: Loss Learning Curves

- **Epochs**: Epochs is the number of times a machine learning model goes through the whole training dataset during the training procedure. After experimenting with 2, 3 and 4 epochs, the following results were produced:
  - **4 epochs:** As we can observe from the curves below, 4 epochs created overfit, since the f1 and loss training scores were increased and the respective validation scores were decreased significantly.

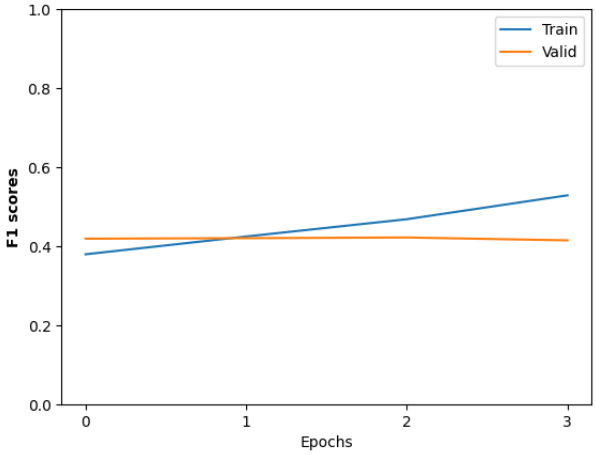


Figure 19: 4 Epochs: F1 Learning Curves

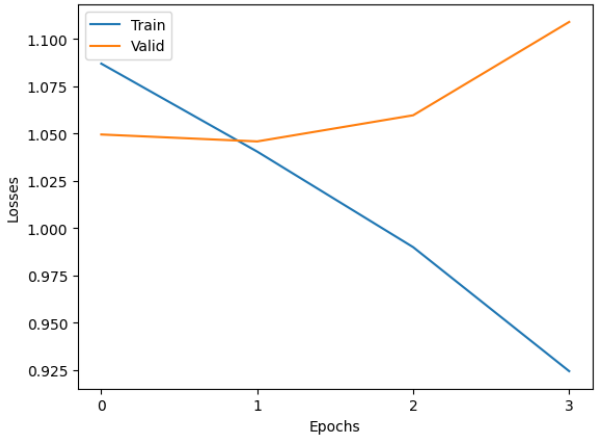


Figure 20: 4 Epochs: Loss Learning Curves

– **3 epochs:** Graphs generated with 3 epochs indicated advance over those with 4 epochs, because a smaller overfit was created. Consequently, I've decided against using 4 epochs in my final model and I chose to explore whether 2 epochs perform better than 3 epochs.

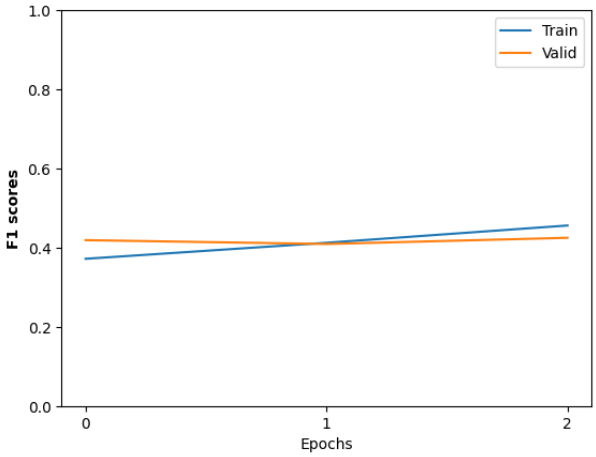


Figure 21: 3 Epochs: F1 Learning Curves

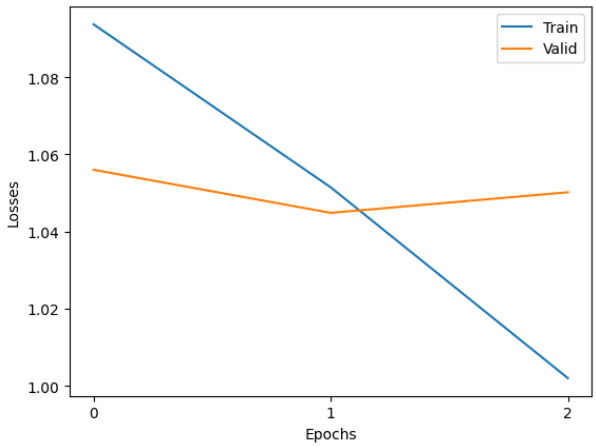


Figure 22: 3 Epochs: Loss Learning Curves

– **2 epochs:** Surprisingly, 2 epochs created the optimal scores, since the f1 validation score reached a ceiling of 0.43 and the validation loss consistently decreased, reaching a minimum of 1.04. Moreover, 2 epochs demonstrated the least overfitting when compared to running 3 or 4 epochs. Considering the aforementioned factors, it became clear that 2 epochs consist the best selection, and it will be implemented in my final GreekBERT model.

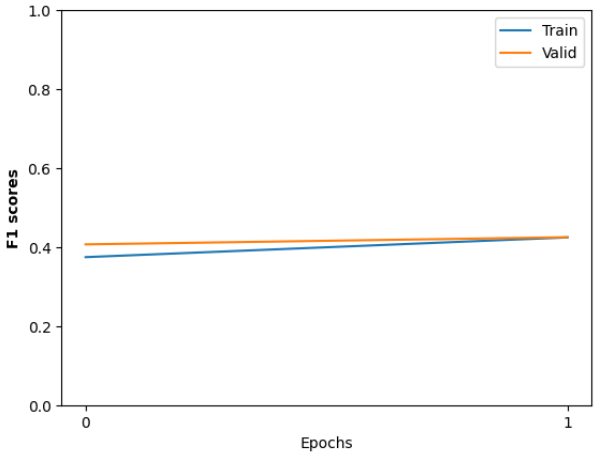


Figure 23: 2 Epochs: F1 Learning Curves

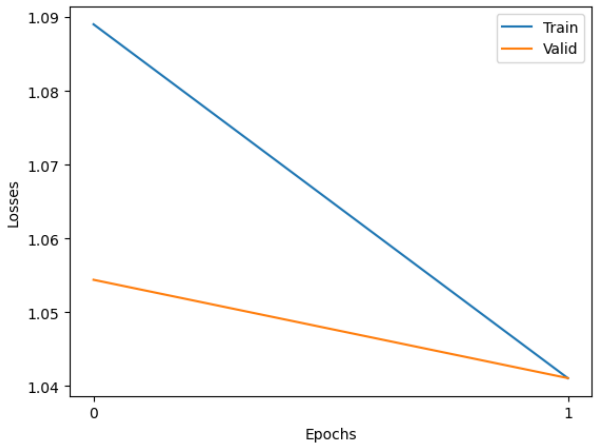


Figure 24: 2 Epochs: Loss Learning Curves

- **Gradient Clipping:** Gradient clipping is a method, which is utilized in the training of a neural network to avoid the **exploding gradient problem**, leading to more stable and reliable networks. The idea behind gradient clipping is very simple. More specifically, it establishes a threshold value and then scales down gradients whose norms exceeds this threshold. As for me, in all the experiments so far the gradient clipping method was employed with a clip value of 1. However, it was necessary to examine how the model would behave without applying this technique. Upon scrutinizing the scores, I noticed that without this method, the results were a bit inferior and overfit was occurred. Therefore, I ended up that it was imperative to retain the method in the model's implementation.

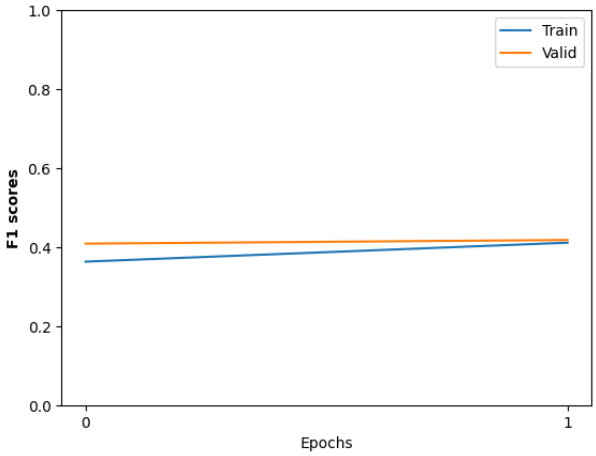


Figure 25: Without Clip: F1 Learning Curves

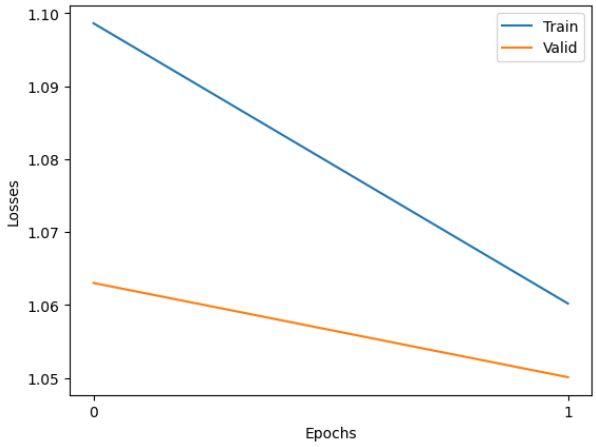


Figure 26: Without Clip: Loss Learning Curves

- **BertModel vs AutoModel:** BertModel and AutoModel refer to different ways of initializing pre-trained models. The difference between them is that AutoModel is a more general class that can be used with various pre-trained models, while BertModel is specifically designed for BERT models. In my experiments and in the final model the AutoModel is used, because I took into consideration the huggingface's documentation. For experimental purposes, I also explored the BertModel to assess whether it could yield superior results, allowing me to compare its performance with the AutoModel, but the results remained exactly the same, rendering further analysis unnecessary.

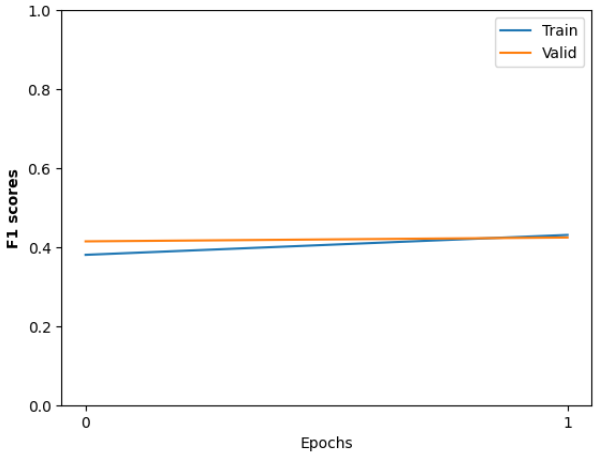


Figure 27: BertModel: F1 Learning Curves

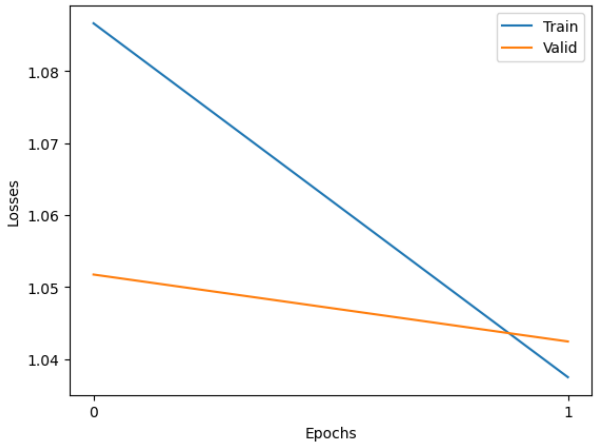


Figure 28: BertModel: Loss Learning Curves

3.1.2. DistilGREEK-BERT model. Afterwards, I conducted the exactly same experiments using the DistilGREEK-BERT model, in order to check if it would lead to better or different results than these produced by the GreekBERT model.

- **Batch size:** I trained the model twice, using batch sizes of 32 and 16, and I noticed that the produced scores were very similar. The only distinction was that with a batch size of 32, the f1 validation score plateaued at 0.39, while it reached 0.41 with a batch size of 16. For this reason, this slight difference served as the criterion for selecting a batch size of 16 over 32.

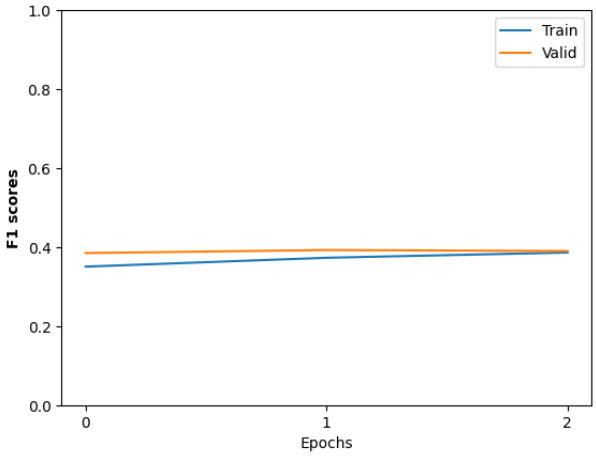


Figure 29: Batch Size 32: F1 Learning Curves

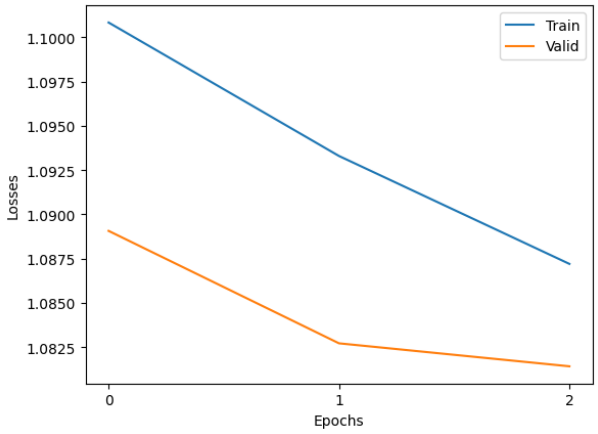


Figure 30: Batch Size 32: Loss Learning Curves

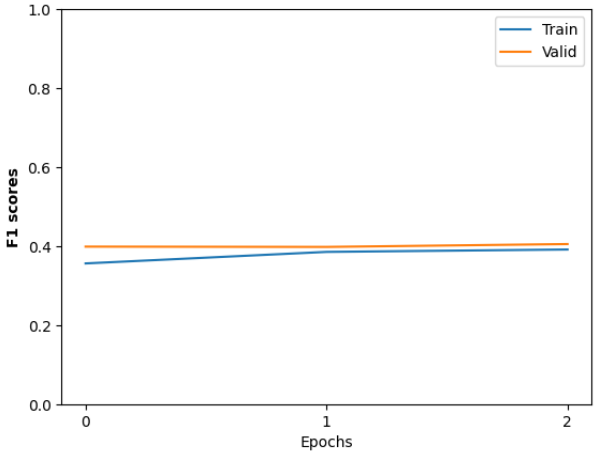


Figure 31: Batch Size 16: F1 Learning Curves

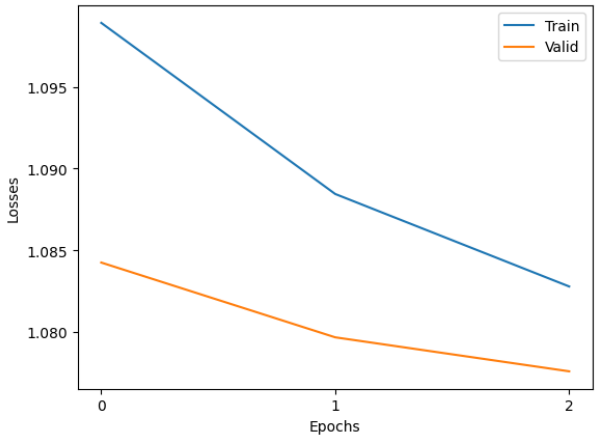


Figure 32: Batch Size 16: Loss Learning Curves

- **Learning Rate:** Similar to my approach with the Greek-BERT model, I experimented with the learning rates 2e-5, 3e-5, and 5e-5. As expected based on the progression of experiments thus far, the outcomes performed a high degree of similarity. Hence, the selection was informed by the f1 validation score. The respective scores for the learning rates 2e-5, 3e-5, and 5e-5 were 0.40, 0.41, and 0.40, so the optimal learning rate was 3e-5.

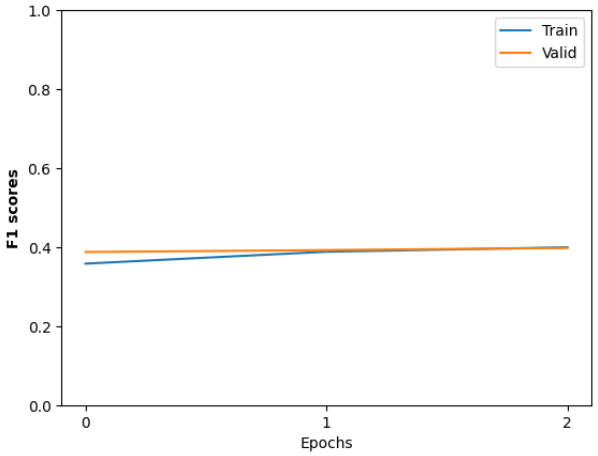


Figure 33: Learning rate 2e-5: F1 Learning Curves

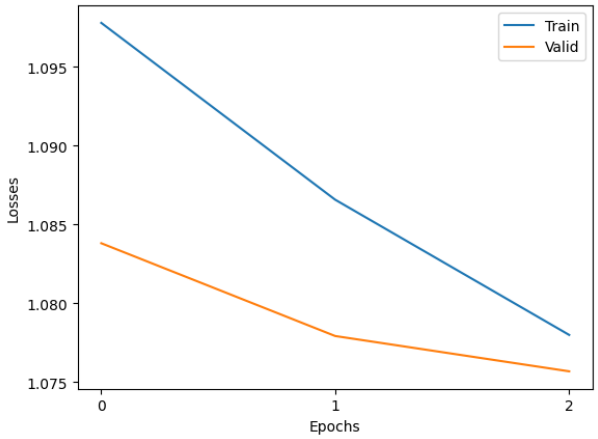


Figure 34: Learning rate 2e-5: Loss Learning Curves

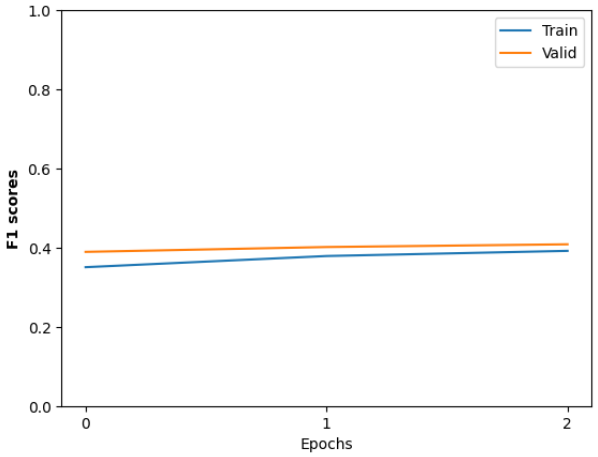


Figure 35: Learning rate 3e-5: F1 Learning Curves

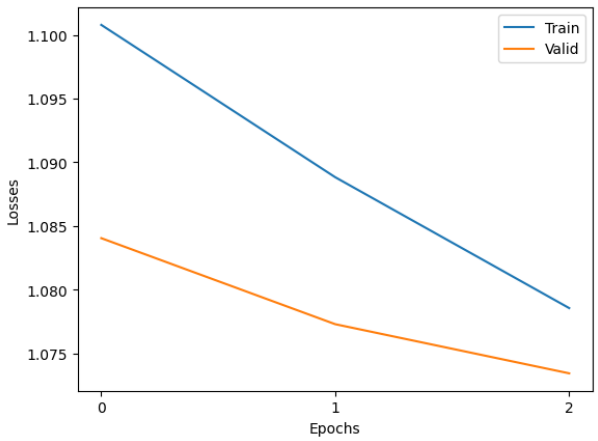


Figure 36: Learning rate 3e-5: Loss Learning Curves

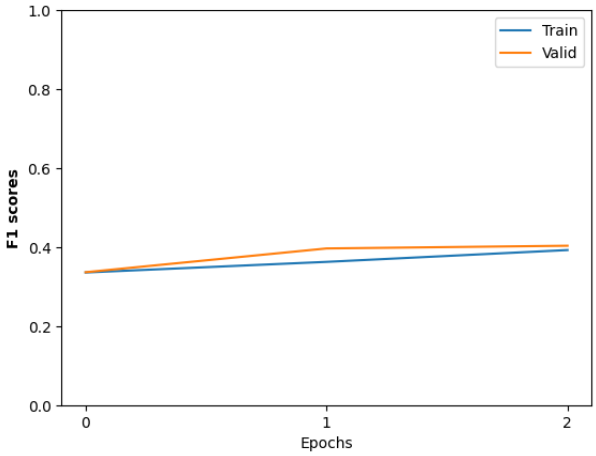


Figure 37: Learning rate 5e-5: F1 Learning Curves

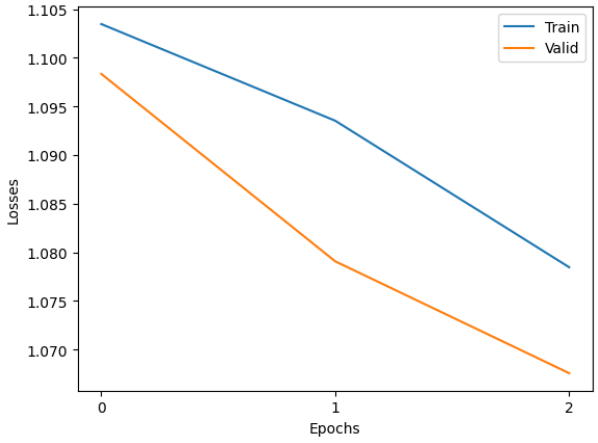


Figure 38: Learning rate 5e-5: Loss Learning Curves

- **Padding and Truncation:** As for the padding, I conducted 3 experiments using 3 different maximum lengths: 60, 100, 140(the maximum number of tokens among all sentences in our datasets). The results were identical, probably due to the bad dataset. Consequently, I opted for a maximum length of 140, as choosing 60 or 100 would result in significant information loss.

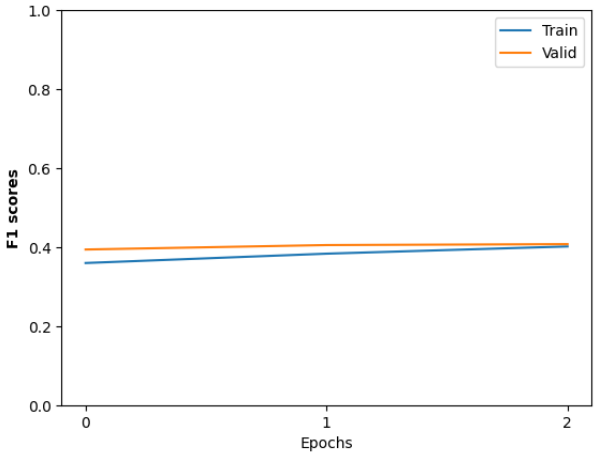


Figure 39: Max Length 60: F1 Learning Curves

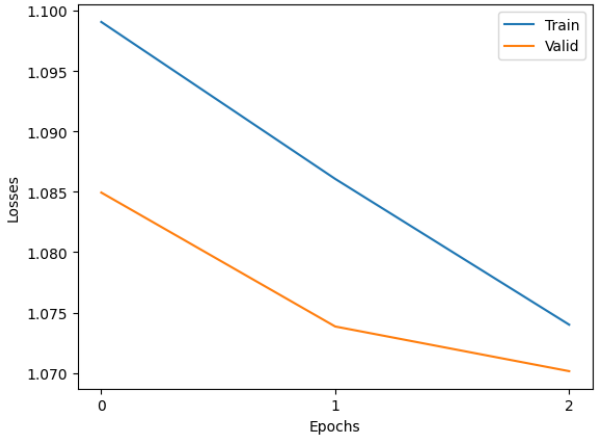


Figure 40: Max Length 60: Loss Learning Curves

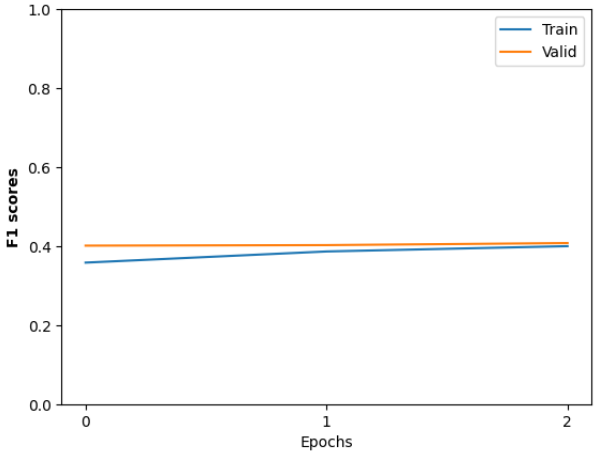


Figure 41: Max Length 100: F1 Learning Curves

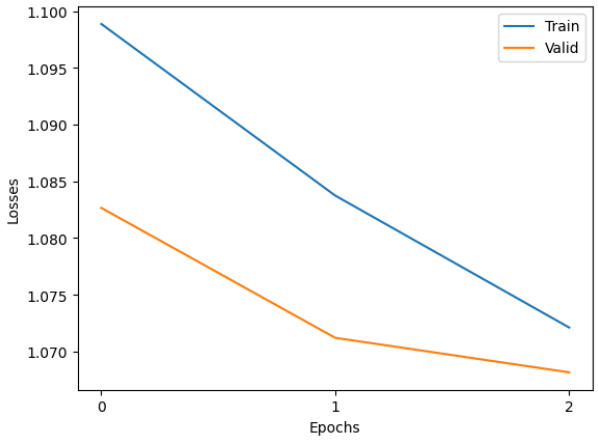


Figure 42: Max Length 100: Loss Learning Curves

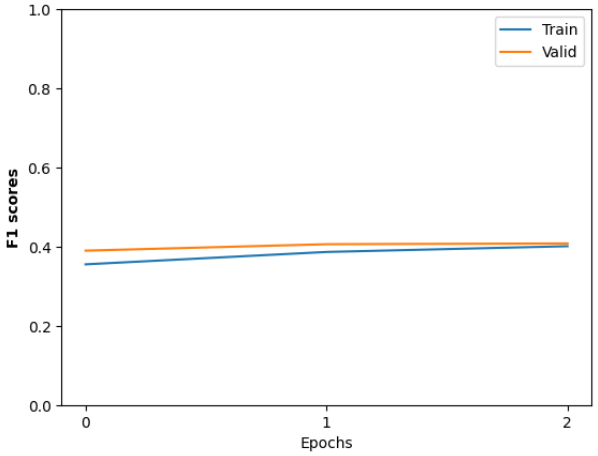


Figure 43: Max Length 140: F1 Learning Curves

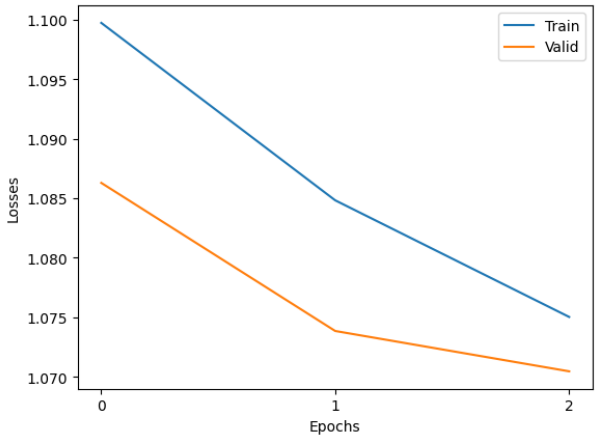


Figure 44: Max Length 140: Loss Learning Curves

- **Epochs:** As I mentioned in the previous model, the epochs among which I experimented were 2, 3 and 4 and the results for each case are the following:

– **2 epochs:** Training the model for only 2 epochs seemed inappropriate for this model, since the training and validation f1 scores set fail to reach the expected ceiling of 0.40-0.41 and remained low around 0.38 and 0.39 respectively.

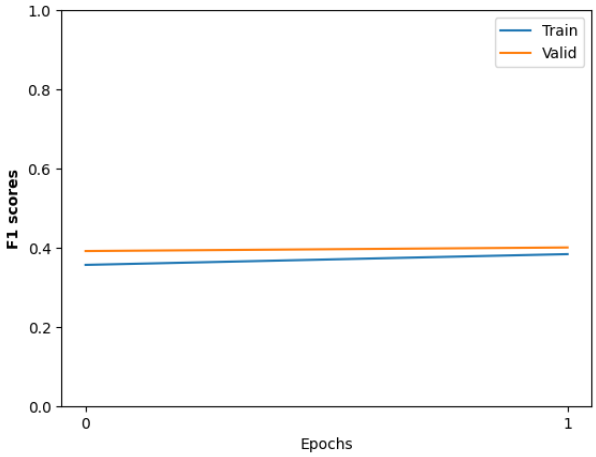


Figure 45: 2 Epochs: F1 Learning Curves

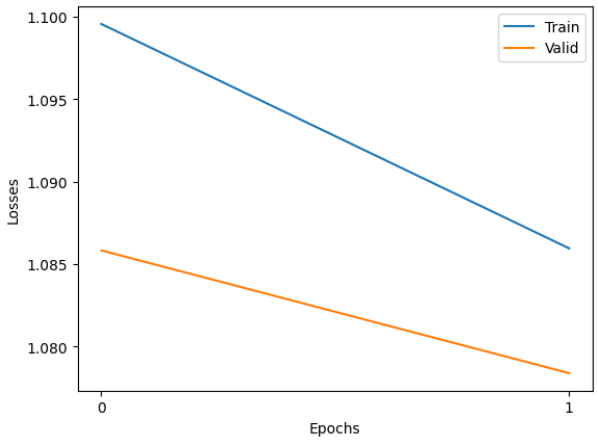


Figure 46: 2 Epochs: Loss Learning Curves

– **3 epochs:** For the whole experimentation so far a three-epoch strategy was employed, because it offers a middle ground between these 3 number of epochs. As previously highlighted, the best results occurred by a three-epoch training were the following: A plateau of 0.40 and 0.41 f1 scores for training and validation respectively, in combination with losses of 1.07. To ascertain potential further improvements, it became necessary to investigate whether extending the training to 4 epochs could yield superior results.

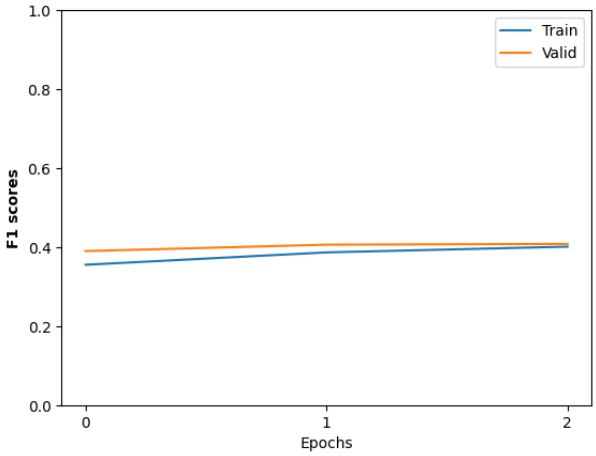


Figure 47: 3 Epochs: F1 Learning Curves

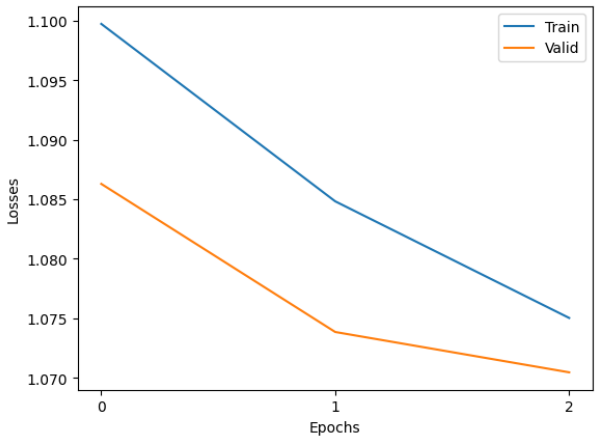


Figure 48: 3 Epochs: Loss Learning Curves

– **4 epochs:** Remarkably, the optimal results were achieved with a training duration of 4 epochs.This phenomenon arises due to the fact that, the f1 training score was increased from 0.40 to 0.41 and both training and validation looses were decreased from 1.07 to 1.06.

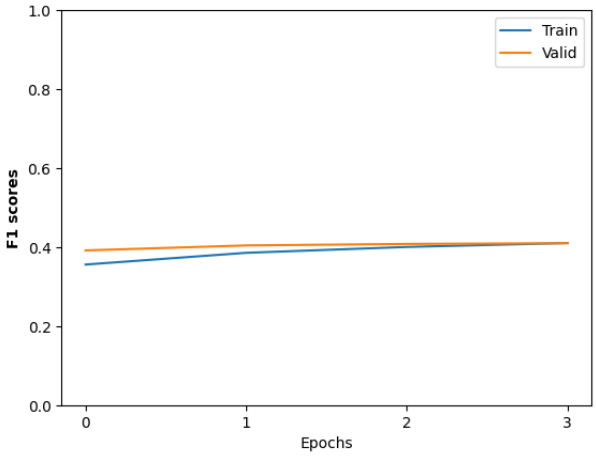


Figure 49: 4 Epochs: F1 Learning Curves

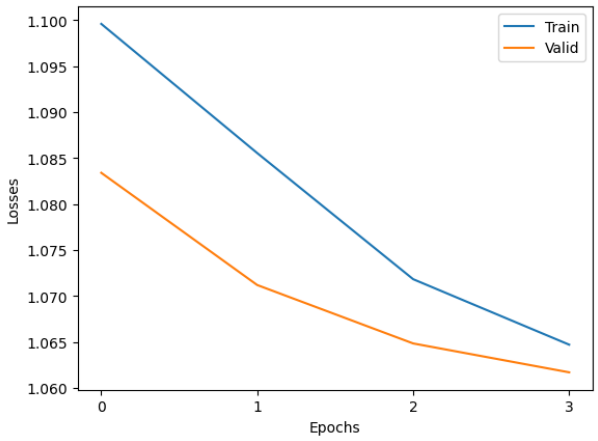


Figure 50: 4 Epochs: Loss Learning Curves

- **Gradient Clipping:** In the previous experiments utilizing the DistilGREEK-BERT model, the gradient clipping method with a threshold value of 1 was employed in order to mitigate the risk of exploding gradients. It is worth mentioning that, when using the same hyperparameters values, the model without the application of this technique exhibited diminished performance. Specifically, the training and validation f1 scores plateaued at 0.34 and 0.36, respectively, accompanied by losses of 1.10, in contrast to the more favorable outcomes observed when employing the gradient clipping method. Consequently, it becomes evident that the implementation of this technique is crucial and should, under no circumstances, be overlooked.

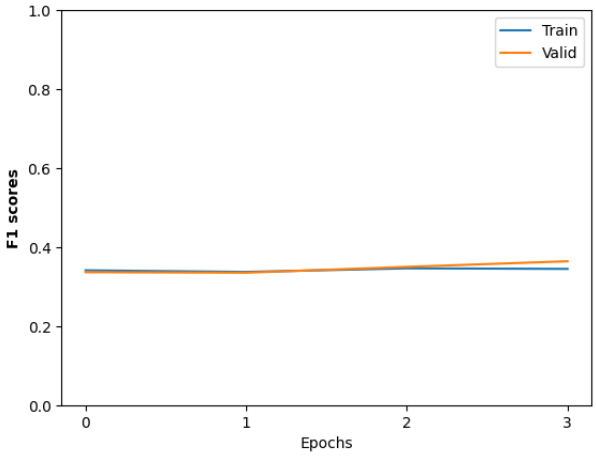


Figure 51: Without Clipping: F1 Learning Curves

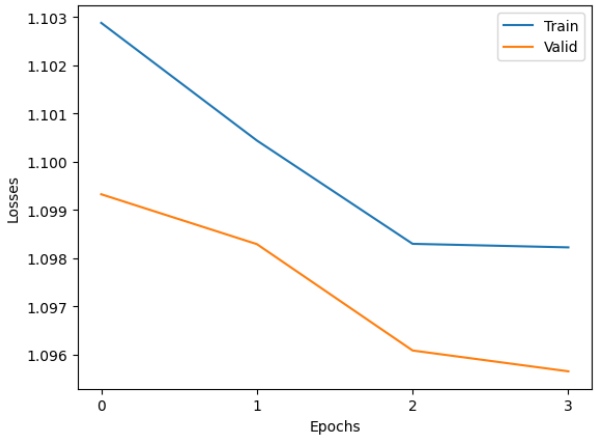


Figure 52: Without Clipping: Loss Learning Curves

- **AutoModel vs BertModel:** We expected that employing the BertModel instead of the AutoModel would end up to similar results, like happened with the GreekBERT model. However, our expectations didn't come true. This difference can be attributed when using the same hyperparameters that had optimized the AutoModel's performance before. Surprisingly, with the BertModel, these hyperparameters led to suboptimal results, evidenced by a training loss of 1.1 and a validation loss of 1.09 in the fourth epoch. Furthermore, the f1 scores remained at a low level, around 0.35. Hence, I concluded that the BertModel is not as suitable as the AutoModel for this sentiment classification task.

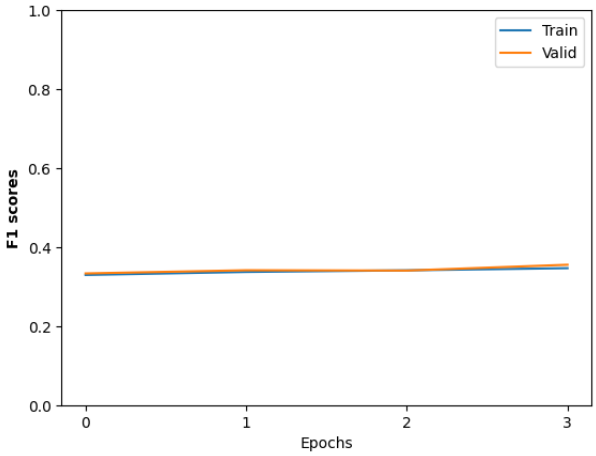


Figure 53: BertModel: F1 Learning Curves

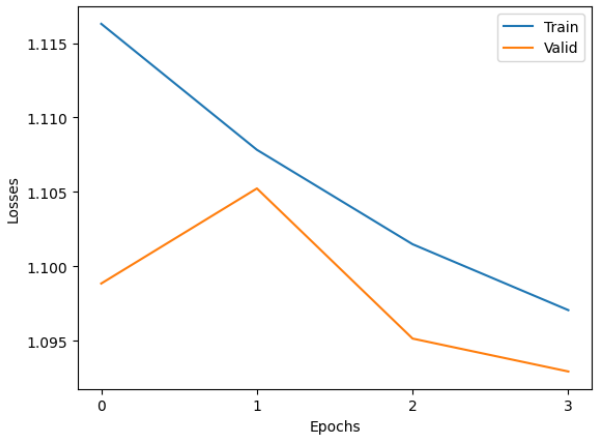


Figure 54: BertModel: Loss Learning Curves

3.2. Hyper-parameter tuning

<Describe the results and how you configured the model. What happens with under-over-fitting??>

Hyperparameters are configurations that influence the behavior of the algorithm, the performance of the model and the size of over and underfitting. In other words, different hyperparameter values can lead to completely different classifier results. In my model, the principal hyperparameters I tried to tune were the batch size, the number of epochs, the learning rate and the maximum length of a sentence. The procedure I followed in order to tune these hyperparameters was the following: I conducted experiments for each one, trying the values that Bert's creators suggest, then I used the optimal hyperparameter value for the following experiments. However, I realized that two models with the exact same configuration do not always yield the same results, so despite the above experimentation, I used the **Optuna framework**, in order to find the consistently better values.

3.3. Optimization techniques

<Describe the optimization techniques you tried. Like optimization frameworks you used.>

For the exercise implementation i used the **Optuna framework**. Optuna is an optimization framework used in machine learning, aimimg to discover the optimal hyperparameters in order to improve the effectiveness of a model. Personally, I used this framework to discover the optimal batch size, learning rate and number of epochs, because as I mentioned above two models with the exact same configuration do not always end up to the same outcomes. It is worth mentioning that, I only tried 15 trials for each model, because as it is obvious more trials would require more time and the usage of the GPU in kaggle is limited. For the best utilization of the framework, I used some of the visualization features optuna provides, in order to analyze the optimization results visually. More specifically:

- **Slice Plot**: From this plot, we can easily understand which hyperparameter values create the best scores. Specifically, for the GreekBERT model, the combination of a batch size of 16, a number of epochs of 2, and a learning rate of 2e-5 seemed ideal. However, the limitation of only 15 trials prevented me from considering this scenario as the best possible. To address this, I compared its results with the current best-performing configuration and I ended up that the outcomes were exactly the same. Consequently, there was no reason to incorporate these hyperparameter values into my final report. In contrast, for the DistilGREEK-BERT model, a batch size of 16, a number of epochs set at 4, and a learning rate of 5e-5 generated a slightly better results compared to the current best-performing combination. Therefore, the hyperparameters values the Optuna framework suggested for this model were beneficial.

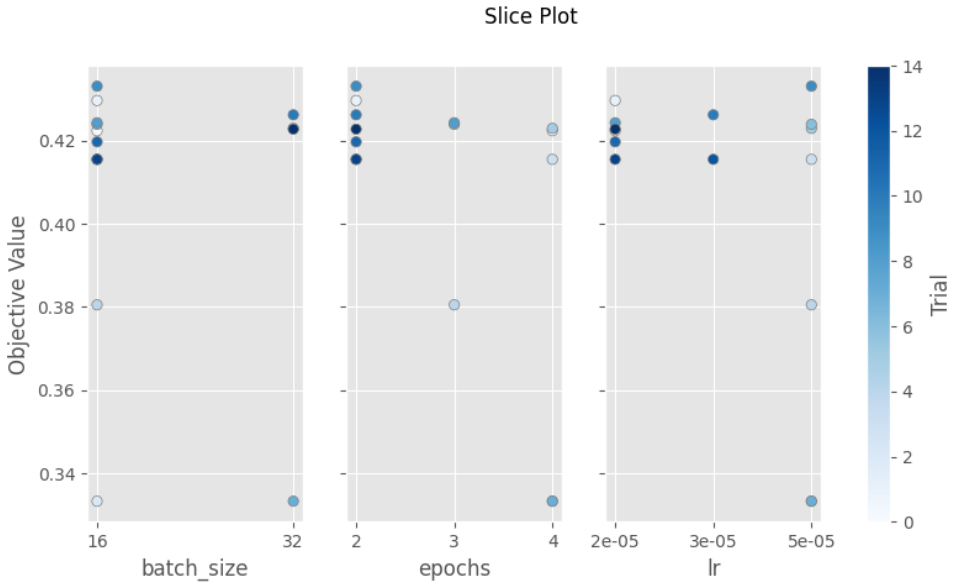


Figure 55: GreekBERT: Slice Plot

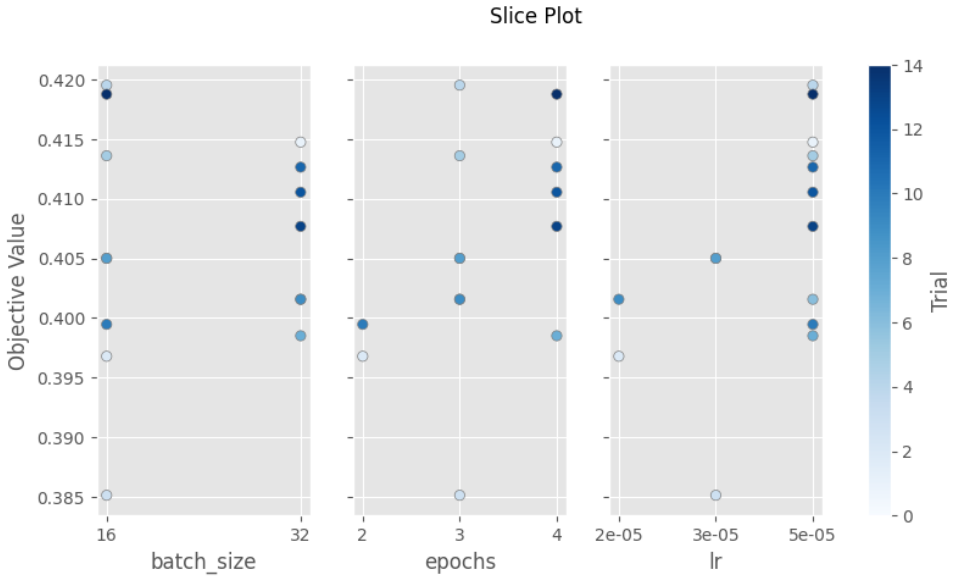
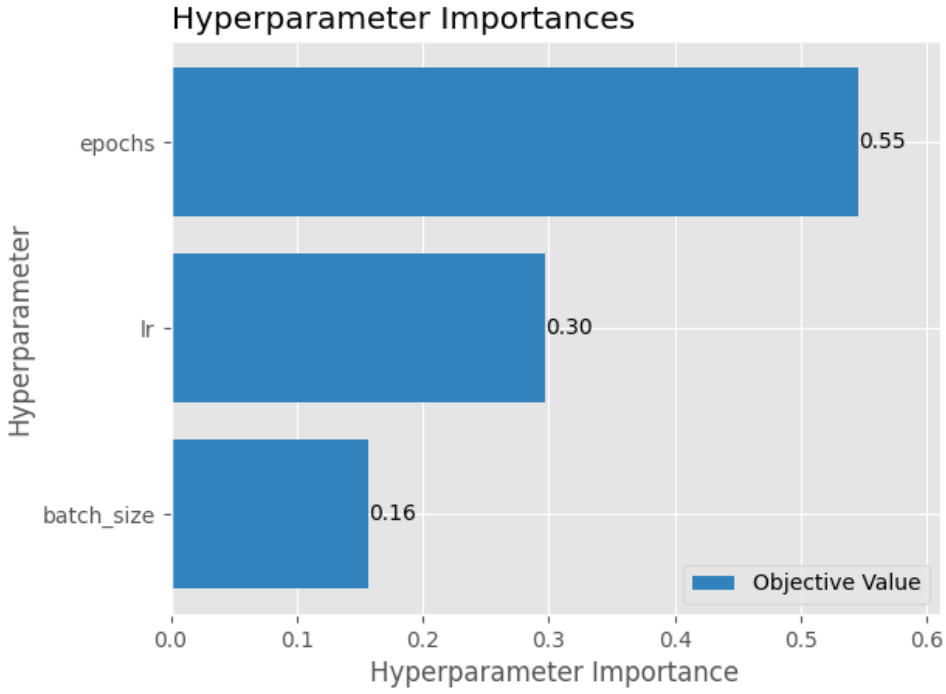
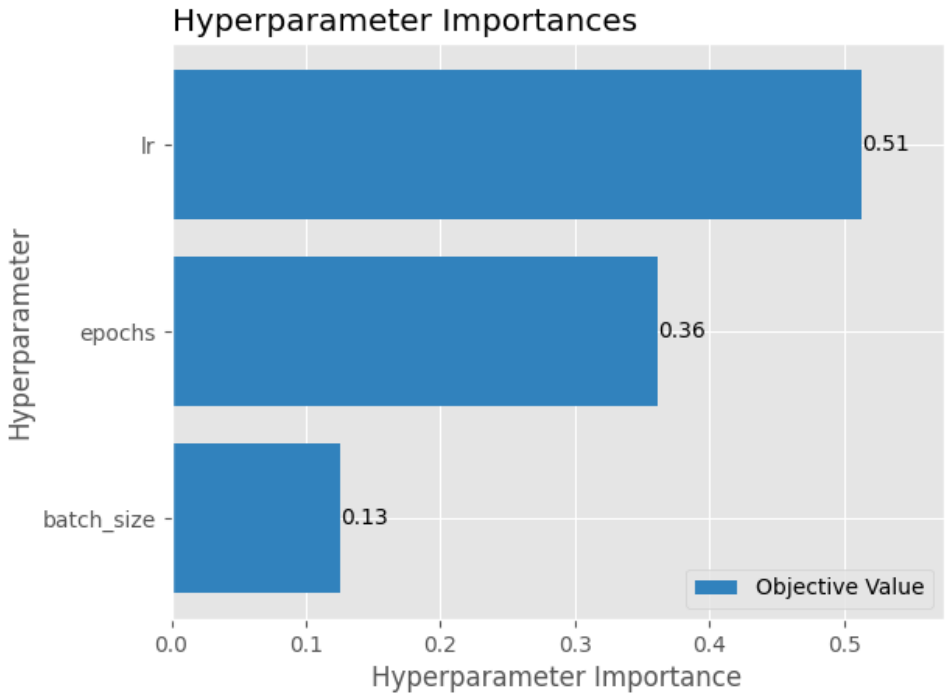


Figure 56: DistilGREEK-BERT: Slice Plot

- **Hyperparameters Importance**: The hyperparameters importances graph refers to the impact of each hyperparameter on the model's performance. The graph reveals that the number of epochs assigned a significant influence of 55 percent for the GreekBERT model, while the learning rate had a modest impact of 0.30 percent. In contrast, for the DistilGREEK-BERT model the learning rate noted an influence of 0.51 percent and the number of epochs a 0.36 percent impact. Last but not least, the batch size had a minor effect on both model, since the assigned importance is 0.16 and 0.13 percent respectively.





Note:If you want to run the Optuna code in my Notebook change the *use_optuna* variable to True.

3.4. Evaluation

<How will you evaluate the predictions? Detail and explain the scores used (what's fscore?). Provide the results in a matrix/plots>
For evaluating my model, I used F1 learning curves, loss learning curves, ROC curves and Confusion matrices.

3.4.1. ROC curve. ROC curve is another evaluating measure, which is created by plotting the true positive rate opposed to the false positive rate at various thresholds, while AUC is the area under the ROC curve. In the final GreekBERT model AUC is around 0.58-0.61, while in the DistilGREEK-BERT AUC is around 0.57-0.60, which means that the ability of both models to predict is better than random guessing, but still remains low.

3.4.2. F-Score. F-score is a classification algorithm's performance metric, that combines precision and recall into a single measure and it ranges from 0 to 1, where a higher value indicates better performance. After the above experimentation, I ended up that the best f1 training and validation scores are 0.43 for the GreekBERT model and 0.42 for the DistilGREEK-BERT.

3.4.3. Loss. A loss learning curve is a graphical representation of a machine learning model's performance over epochs and depicts the difference between the predicted values and the actual values. In my final models, i used the CrossEntropy loss function

leading the training and validation losses into a decrease about 1.074 for the Greek-BERT and 1.076 for the DistilGREEK-BERT model and this reveals that both models have a similar performance to seen and unseen data.

3.4.4. Confusion matrix. A confusion matrix represents the prediction's results in matrix form and it shows how many predictions are correct and incorrect on a validation or a test set. Taking into consideration the confusion matrices below, we notice that the GreekBERT model can predict correct 50 percent of negative sentiments and 49 percent of positive, but struggles to identify the neutral elements, since it has an accuracy of 28 percent. Similar results creates the DistilGREEK-BERT model too, since the accuracy is 57 percent for negative, 38 percent for positive and 28 percent for neutral. In other words, what we conclude from the above outcomes is that all the results and especially the neutral ones are below the desired limit.

Note: In my paper, you can find the ROC Curves and Confusion matrices plots only for my final models, because incorporating the diagrams of all the experiments into my paper would create a sense of complexity making it tedious to read.

4. Results and Overall Analysis

4.1. Results Analysis

<Comment your results so far. Is this a good/bad performance? What was expected? Could you do more experiments? And if yes what would you try?>
In conclusion, the best accuracy the **GreekBERT** model can reach is about 0.43, while the optimal accuracy for the **DistilGREEK-BERT** is 0.42. This is because, a significant number of Sentiments in the dataset is wrong. However, the goal of the Project was to experiment with various techniques and hyperparameters and familiarize with the usage of these pre-trained models. Obviously, there are many experiments, I don't present in this paper or in the Notebook. For example, I could explore the model's performances without using the learning rate scheduler, or alternatively, I could experiment with multiple clip values rather than just one.

4.1.1. Best trial. <Showcase best trial>
Given all the experiments, I ended up that:

- The best trial for the **GreekBERT** model was the one that combines the following: 32 batch size, 3e-5 learning rate, 2 epochs and 140 maximum length, because it created the best results at the best time.

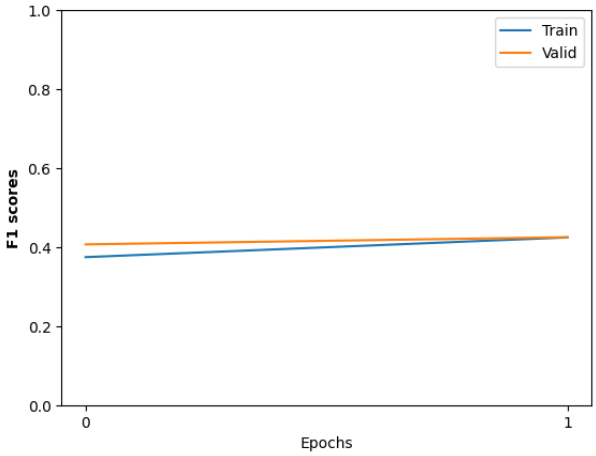


Figure 57: Best GreekBERT model: F1 Learning Curves

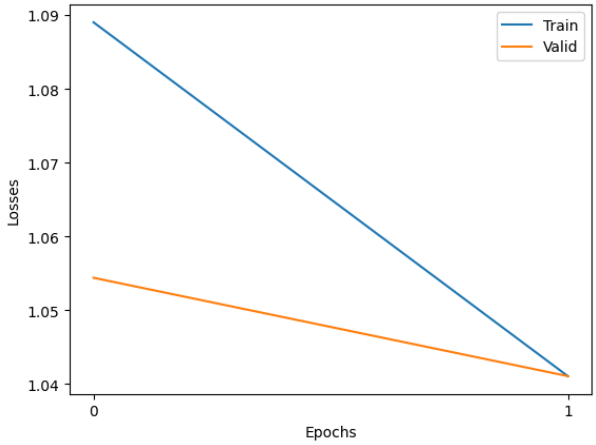


Figure 58: Best GreekBERT model: Loss Learning Curves

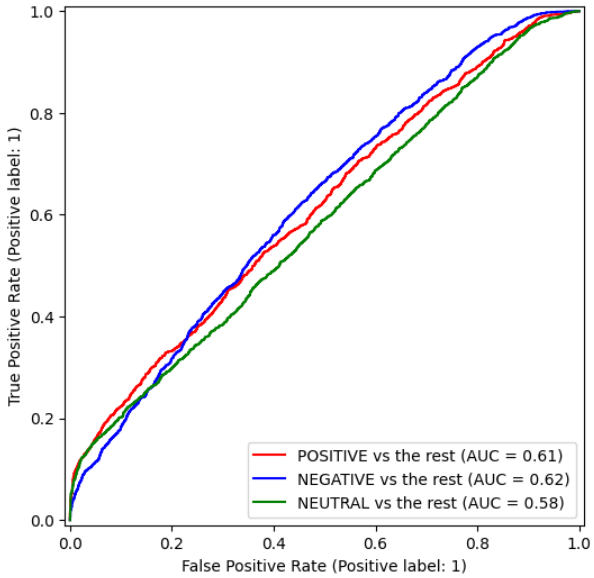


Figure 59: Best GreekBERT model: ROC Curves

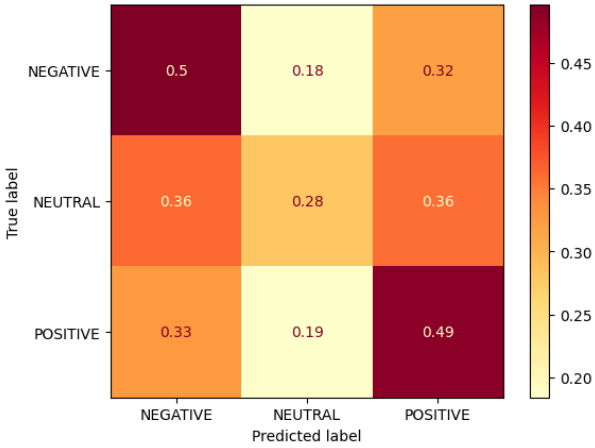


Figure 60: Best GreekBERT model:Confusion Matrices

- Similar, for the **DistilGREEK-BERT** model the best hyperparameters combination is: 16 batch size, 5e-5 learning rate, 4 epochs and 140 maximum length.

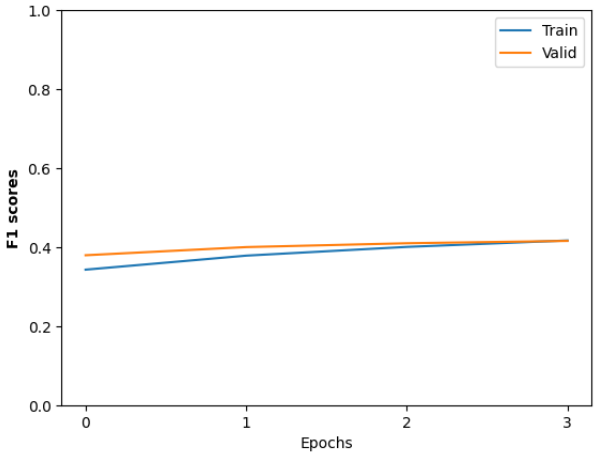


Figure 61: Best DistilGREEK-BERT model: F1 Learning Curves

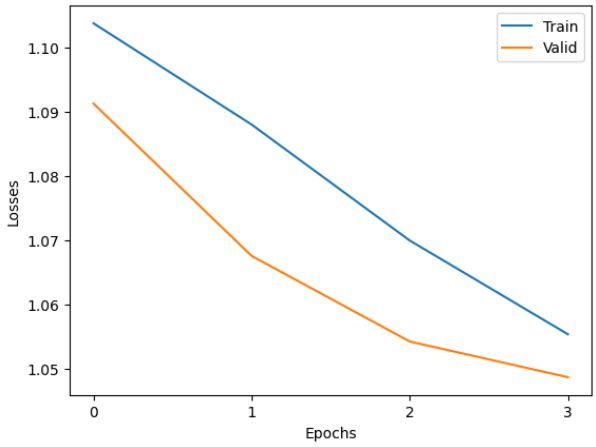


Figure 62: Best DistilGREEK-BERT model: Loss Learning Curves

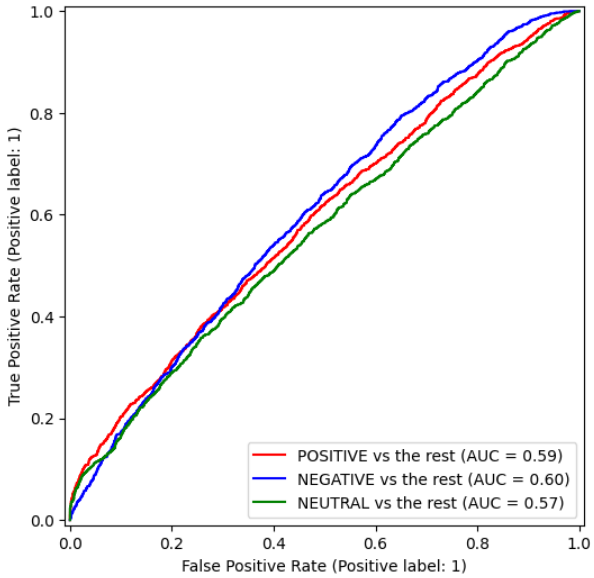


Figure 63: Best DistilGREEK-BERT model: ROC Curves

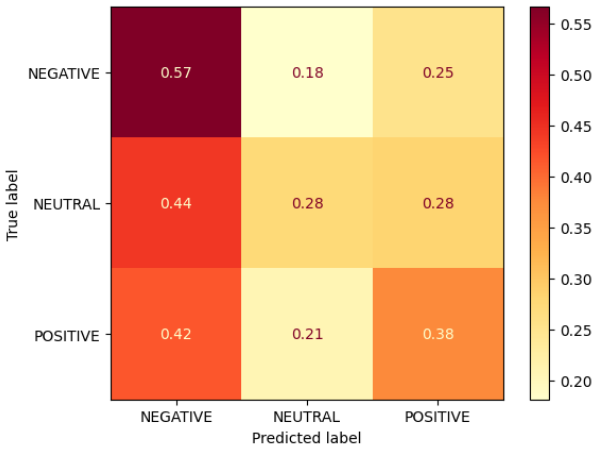


Figure 64: Best DistilGREEK-BERT model:Confusion Matrices

Among the models, GreekBERT stands out as the superior choice, reaching a 0.43 accuracy and a 1.04 loss. In comparison, the DistilGREEK-BERT model achieved a slightly lower 0.42 f1 score in combination with a loss of 1.05. For this reason, the main notebook submission incorporates the results derived from the GreekBERT model.
Note: I created a second submission named *submision_distil.csv*, which contains the final submission for the DistilGREEK-BERT model.

4.2. Comparison with the first project

<Use only for projects 2,3,4>
<Comment the results. Why the results are better/worse/the same?>
This approach is quite better than that of the first assignment. First of all, both pre-trained models are designed to understand the context of words in a sentence, in contrast to logistic regression classifiers. Moreover, BERT is a deep neural network with a huge number of parameters, which give it the ability to learn complex relationships in

data, while logistic regression is simpler and may struggle to capture complicate patterns. The aforementioned excellence can also be seen in the results. In the first project the model corresponded well to seen data, but it had low performance to unseen ones, creating overfitting ,in contrast to this assignment where the model reacts to both in similar way.

4.3. Comparison with the second project

<Use only for projects 3,4>
<Comment the results. Why the results are better/worse/the same?> This assignment closely resembles the second project (the pre-trained models are a bit better), because all three neural networks create similar results. However, in theory, the pre-trained models were expected to improve the model more significantly than deep neural networks, because they are capable to capture long-term dependencies effectively. The reason why these models didn't enhance the model isn't clear. The bad dataset is probably the primary factor contributing to the outcome, but we can't substantiate this estimation with certainty.

4.4. Comparison with the third project

<Use only for project 4>
<Comment the results. Why the results are better/worse/the same?> Both DistilGREEK-BERT and GreekBERT outperform stacked RNNs, creating superior results with f1 scores surpassing 0.40. However, training with stacked RNNs shows similarities to training with pre-trained models, since examining the f1 and loss learning curves, we observe that all three models followed a similar learning trajectory from the input data. This happenned because, despite that BERT and stacked RNNs are two different architectures, they have some similarities in how they capture sequential information from the input data. Another point worth mentioning is that Bert training is slower than stacked RNNs training, because Bert's architecture is more complicated and more computation is required.

5. Bibliography

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

<You should link and cite whatever you used or "got inspired" from the web. Use the / cite command and add the paper/website/tutorial in refs.bib>
<Example of citing a source is like this:> [1] <More about bibtex>

REFERENCES

- Sklearn Documentation: <https://scikit-learn.org/stable/>
- Emoji Removal: <https://gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b>
- PYTORCH documentation: <https://pytorch.org/docs/stable/index.html>

- BERT Fine-Tuning Tutorial with PyTorch: <https://mccormickml.com/2019/07/22/BERT-fine-tuning/>