UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Kleopatra Karapanagiotou*
sdi: lt12200010

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

Content

1. Abstract

The purpose of the 4th and last assignment is to develop a multi-class sentiment classifier for a dataset containing tweets about the Greek general elections, by finetuning two pretrained BERT models, the GreekBERT and the DistilGREEK-BERT. Steps on data preparation, experiments and hyperparameter tuning are thoroughly discussed in the following sections. An overall analysis and comparison of the results of all projects on this dataset are discussed in the final sections.

2. Data processing and analysis for BERT

2.1 Pre-processing

The choice of the text preprocessing steps was aligned with the version of Greek BERT model chosen for the assignment. In our case bert-base-greek-uncased-v1 was chosen. This version expects the input text to be lowercased and deaccented. Even though the default tokenizer takes care of these steps, they were explicitly applied during preprocessing to make sure the tokenizer input is in the correct format in advance. Furthermore, the pretraining corpora for GreekBERT included full sentence text from multiple resources of Greek[1]. Condsidering the above, the preprocessing steps in our tweets were adjusted as follows:

		train	val	test
Cleaning	• Duplicate rows removal • Null-values removal	✓	✓	✗
Preprocessing	Step 1 • Removal of common characters found only in tweet content: RT-tags (retweets), @unsernames, urls , html and http links Step 2 • Text normalization for deaccenting • Punctuation separation and removal • Latin characters removal • Number removal Step 3 • Lowercasing • Emoji removal • Removal of extra whitespace	✓	✓	✓
Post-processing	• Removal of NaN rows and updating of the index in the whole dataframe , in case any row left empty after the preprocessing applied.	✓	✓	✗
Column extraction	• Extraction of Text and Sentiment columns/ Extraction of text column only • Sentiment Mapping {'NEUTRAL': 0, 'POSITIVE': 1, 'NEGATIVE': 2}	✓	✓	✓

[1] https://huggingface.co/nlpaueb/bert-base-greek-uncased-v1

New dataframe generation	• New cleaned dataframes containing only the preprocessed text column and the encoded sentiment column.	✓	✓	✓

2.2 BERT Tokenizer

In order to apply the pre-trained BERT, we must use the tokenizer provided by the hugginface library. This is because:
(1) the model has a specific, fixed vocabulary of 35000 tokens and
(2) the BERT tokenizer has a particular way of handling out-of-vocabulary words, through subword tokenization with the WordPiece algorithm.
The function **preprocessing_for_bert()** incorporates all the steps for BERT tokenization:

1. Takes as input an array of texts to be processed.
2. Creates empty lists for the input ids and the attention masks.
3. Iterates through each text(tweet) in the array of texts and tokenizes the tweet with the tokenizer.enode_plus function, which combines the following steps:
    - Splits the sentence into tokens.
    - Adds the special [CLS] and [SEP] tokens.
    - Maps the tokens to their Ids, by convert them into indexes of the tokenizer vocabulary,
    - Pads or truncates all sentences to the same maximum length.
    - Creates the attention masks which explicitly differentiate real tokens from [PAD] tokens.

    The input id and the attention mask of each encoded sentence is then appended to the empty lists defined above, which are then converted into tensors. The input id tensor contains the token ids to be fed into the BERT model and the attention mask tensor contains the indices specifying which tokens should be attended by the model.
4. The input_ids and attention_mask tensors are returned by the function.

For specifying the maximum sentence length in tokens, the train and validation datasets were concatenated and gave a maximum sentence length of **95 tokens**, which was used as the max_len value in the tokenization.
Finally, the labels in the Sentiment column of the train and validation datasets are also converted into torch tensors. The transformation of the labels list into tensors is not incorporated in the preprocessing_for_bert function, to be able to use the function in the test dataset, which contains no column for Sentiment labels.

2.2.1 Notes on data preprocessing on the test set

During preprocessing of test data for BERT the following TypeError occurred:
TypeError: TextEncodeInput must be Union[TextInputSequence, Tuple[InputSequence, InputSequence]] and based on this Github post the main root cause was that after applying the initial text preprocessing steps, there were two rows in the test dataset left empty.

```
{'New_ID': 1045, 'Text': '#kinal #andreastzouramanis #politiallios #ekloges
https://t.co/1sjjklKPS0νδρέας-τζουραμάνης-στο-ράδιο-γάμμα/', 'Party': 'PASOK'}
{'New_ID': 1911, 'Text': 'https://t.co/V1rrmvJNDhοι-υποψήφιοι-της-νδ-περιόδευσαν-στους/',
'Party': 'ND'}
```

Considering the content of the above two tweets (1045, 1911) it can be observed that the tweets are left empty due the following preprocessing steps: hashtags, Latin characters and https link- removal, meaning that these two tweets contain only hashtags, links and no natural language content, which can help us identify a sentiment label, not even through human inference. The original preprocessing

was kept as such, meaning that tweets containing latin characters, hashtags or links are not considered emotionally connotated. Especially for the Latin character removal it was my decision to remove them because if not removing, the tweets would contain additional noise like : boiler plate words, e.g: video,photo,page,skaitv and many other sentiment-irrelevant content.

2.3 Analysis

An example sentence before and after encoding the tweet and applying the tokenization process described above can be seen below:

```
Original:  απολυμανση κοριοι απεντομωση κοριος απολυμανσεις κοριος την
κυριακη κουλης τσιπρα κοριοι απολυμανση καταπολεμηση κοριων απεντομωση
για κοριους

Encoded tweet:['[CLS]', 'απολυμανση', 'κορ', '##ιοι', 'απε', '##ντο',
'##μω', '##ση', 'κορ', '##ιος', 'απολυμανσει', '##ς', 'κορ', '##ιος',
'την', 'κυριακη', 'κουλ', '##ης', 'τσιπρα', 'κορ', '##ιοι',
'απολυμανση', 'καταπολεμηση', 'κορ', '##ιων', 'απε', '##ντο', '##μω',
'##ση', 'για', 'κορ', '##ιους', '[SEP]']

Token IDs:[101, 24023, 1643, 4314, 1294, 2108, 4651, 506, 1643, 2251,
31429, 278, 1643, 2251, 350, 933, 9269, 415, 2269, 1643, 4314, 24023,
4109, 1643, 781, 1294, 2108, 4651, 506, 353, 1643, 7418, 102, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

2.4 Data partitioning for train, test and validation

In our experiment setting our data are already partitioned on 3 .csv files, train, validation, and test. No further partitioning was performed. Training data were used for finetuning the pretrained GreekBERT and DistilGreekBERT models, validation data were used as the evaluation part of the finetuned models and the plotting of learning curves and evaluation metrics. And finally test data were used to predict the polarity of each tweet by creating a new column named: Predicted and adding the predictions of our finetuned classifier.

Before feeding our data for finetuning some additional dataset specific steps had to be applied:

<table>
<tr><td colspan="2">Combine the training/validation inputs (tensors of input ids, attention masks and labels) into a TensorDataset.</td><td>Combine the test inputs (tensors of input ids, attention masks) into a TensorDataset.</td></tr>
<tr><td colspan="3">Select a batch size value (16 or 32 are recommended)</td></tr>
<tr><td colspan="3">Create an iterator for each dataset (train/val/test) using the Pytorch DataLoader class. This will help save on memory during training and boost the training speed.</td></tr>
<tr><td colspan="2">RandomSampler() for train dataset to process random batches. During training we do not really care about the batch order.</td><td>SequentialSampler() for validation and test dataset, to not change the order of the batches and be able to concatenate them for predictions.</td></tr>
</table>

3 Algorithms and Experiments

During fine tuning experiments I used two general purpose language models for the Greek language, the Greek-BERT and the DistilGreekBERT. Greek-BERT is the monolingual Greek version of the original BERT model with 112 million trainable paameters. It uses the same architecture as BERT-base-uncased (12 Encoder layers each of which has 12 self-attention heads and hidden size 768), was pretrained on 29 GB of Greek text and the model's vocabulary contains a total of 35.000 tokens. The idea behind the creation of DistilGreekBERT was to have a significantly smaller model that will take less time to train and at the same time will retain a good percentage of the bigger model's (GreekBERT) performance. To achieve this, the author of DistilGreekBERT (Karavangeli, E. 2023) used a technique for Transfer Learning named Knowledge Distillation. Based on this technique the bigger model (in our case GreekBERT) acts as the teacher model that supervises the student model (DistilGreekbert) during pretraining, so that the student gains knowledge through the teacher. With this compression technique a 40% smaller model with 70 million parameters (6 Transformer encoder layers with 6 attention heads) was trained, which managed to retain 96-97% of GreekBERT's performance with an average of 60% faster training in all conducted experiments (PoS,NER,NLI).

Considering the above, my approach during experiments was to apply the same experiment settings in both pretrained GreekBERT and DistilGreekBERT models and compare their results in terms of training time per epoch and evaluation metrics (weighted avg precision, recall,f1 score and accuracy).

3.1 Experiments

All experiments were conducted on a T4-Tesla GPU.
Experiments 1-3: First experiment with the number of epochs (2,3,4), by keeping batch size and learning rate stable (batch_size:32,lr:5e-5)
Experiments 4-6: Experiment with the number of epochs (2,3,4), by using a smaller batch size and same learning rate (batch_size:16,lr:5e-5)
Experiments 7-8: Based on the best number of epochs and batch size that were found from the first 6 experiments, two additional experiments were conducted in order to explore the effect of the learning rate in during training. The learning rate values were chosen out of the recommended values by the original paper[2]

Based on experiments 1-6 and evaluation metrics, for the GreekBERT the optimal number of epochs was 2 and batch size of 32, whereas for the DistilGreekBERT the optimal number of epochs was 4, with a batch size of 16

Overall, the experiments can be summarized in the following:

Experiments	Batch_size:32			Batch_size:16		
	Exp_1	Exp_2	Exp_3	Exp_4	Exp_5	Exp_6
	2_epochs	3_epochs	4_epochs	2_epochs	3_epochs	4_epochs
Learning rate : 5e-5						

[2] https://arxiv.org/pdf/1810.04805.pdf

Experiments on learning rate based on best epoch and batch size number for each pretrained model

Model	GreekBERT	Batch_size:32		DistilGreekBERT	Batch_size:16	
		Exp_7	Exp_8		Exp_7	Exp_8
		lr: 3e-5	lr: 2e-5		lr: 3e-5	lr: 2e-5
Experiments	2_epochs				4_epochs	

Average finetuning time in seconds/per epoch	
GreekBERT	600 (~ 10 MINUTES)
DistilGreekBERT	300 (~ 5 MINUTES)

3.1.1 Functions and Classes during experiments

Class that constructs the Bert-base model:

- **BertClassifier(nn.Module)** contains two additional linear layers on top of the pretrained GreekBERT architecture, and a ReLU activation function between the two linear layers. The dimensionality of the input, hidden and output layers are the following:
  D_in:768 (Which is the embedding vector of size 768 from the [CLS] token)
  Hidden: 50 (Was chosen based on a tutorial for finetuning BERT[3] and kept due to reasonable output results. No experimentation was applied on the size of the hidden layer)
  D_out:3 (For the 3 classes)
  The class is also initialized with the **freeze_bert** parameter which takes a boolean value, indicating whether the pretrained weights of BERT will be updated. In other words, it specifies, whether finetuning will be applied or not. The default value is False, so that finetuning is applied.
  In the **forward**(self, input_ids, attention_mask)**function**: the tensors for input ids and attention masks are fed into the pretrained BERT model, then the last hidden state of the token [CLS] is extracted for the classification task. Finally, the last hidden state is fed as input into the classifier and the logits for each class are computed and returned.

Both pretrained models and tokenizers were called with the same class Automodel/AutoTokenizer:

```
model_name = "nlpaueb/bert-base-greek-uncased-v1"
distil_model_name = "EftychiaKarav/DistilGREEK-BERT"

model=AutoModel.from_pretrained(model_name) #or distil_model_name
tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case = True)
```

[3] https://skimai.com/fine-tuning-bert-for-sentiment-analysis/

```
BertClassifier(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(35000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (classifier): Sequential(
    (0): Linear(in_features=768, out_features=50, bias=True)
    (1): ReLU()
    (2): Linear(in_features=50, out_features=3, bias=True)
  )
)
```

Full GreekBERT-base-uncased-v1 architecture+ custom classification layer

```
BertClassifier(
  (bert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(35000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (classifier): Sequential(
    (0): Linear(in_features=768, out_features=50, bias=True)
    (1): ReLU()
    (2): Linear(in_features=50, out_features=3, bias=True)
  )
)
```

Full DistilGreekBERT architecture+ custom classification layer

Functions for model training and evaluation (training and validation datasets):

- **train(model, train_dataloader, val_dataloader, epochs=4, evaluation=False):** takes as parameters a defined model, the train and validation dataloaders, the number of epochs and the evaluation mode. Initializes empty lists for train and validation loss and accuracy. Iterates over each epoch and in each epoch, it iterates over each batch of the train dataloader. It loads each batch in the GPU if available, zeroes out gradients and performs a forward pass to get the model logits. It computes the train loss of the batch and accumulates the loss of each batch to get the total loss. Then it performs backward pass to calculate gradients and updates model parameters based on the gradients, the optimizer and the learning rate scheduling. Also, it clips the gradients to 1.0 in order to prevent gradient explosion. Then it takes the maximum value out of the logits and calculates the number of correct predictions in the batch along with the total predictions in the batch. Finally, it calculates the average training loss and accuracy for the epoch and appends the values into the corresponding empty lists defined at the beginning. Finally, it calculates the elapsed time for training in the epoch.
  If the evaluation mode is set to True, the evaluation loss and accuracy for the epoch are calculated for the validation dataloader through the **evaluate** function.
  For each epoch the function prints out the
  - Train Loss and Accuracy along with the elapsed time (in seconds)
  - Validation Loss and Accuracy (if evaluation=True)

and plots the training and validation loss over epochs in one subplot and the training and validation accuracy over epochs in another subplot.

- **Initialize_model**(bert_model_name,epochs=4) takes as parameters the Bert model name (either GreekBERT or DistillGreekBERT) needed to instantiate the Bert Classifier class and the number of epochs for calculating the total number of training steps based on the calculation : (count of training instances/batch size)*epochs. Additionally, the function defines and returns the optimizer (Adam, with a specific learning rate and epsilon value), the loss function (Cross entropy loss) and the learning rate scheduler linear warmup(which creates a schedule with a learning rate that decreases linearly from the initial lr set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial lr set in the optimizer[4].
- **bert_predict_val(model, val_dataloader):** takes as parameter the finetuned BERT classifier and the validation dataloader in order to compute the classification report and confusion matrix from the model's predicted and true labels
- **bert_predict_test(model, test_dataloader):** takes the finetuned model and the test dataloader nd returns the probabilities for each [0:neutral, 1:positive, 2:negative]. With a numpy argmax function we get the maximum probability class for each instance of the test set.

Functions used for metric calculation:

- **accuracy(preds, y_true) functon:** takes as parameters the predictions of labels of the model and the true labels. It changes the predictions into one dimensional tensors and calculates the accuracy. This function is specifically used in the train and evaluate functions to calculate and print the accuracy of each epoch.
- **Show_confusion_matrix(confusion_matrix):** takes a confusion matrix as input and creates a heatmap using seaborn's heatmap function.
- **calculate_metrics(y_true, preds) function:** takes as parameters the y true values and the predicted values. It calculates all the metrics asked for this assignment, namely the precision, F1 score and recall. It also returns the estimated the accuracy of the models and prints the classification report as a general summary of all the metrics and the confusion matrix to visualize the quality of the finetuned classification model.

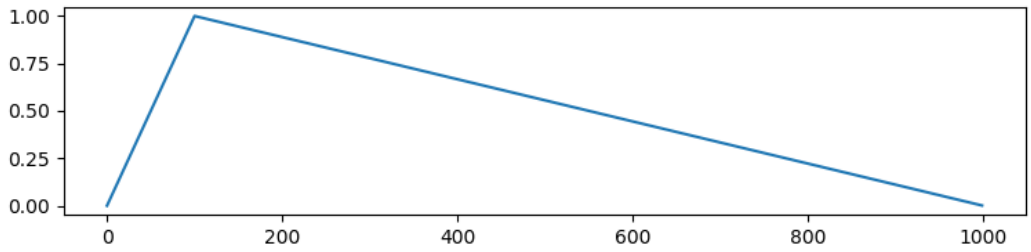
[4] https://huggingface.co/docs/transformers/main_classes/optimizer_schedules

3.2 Hyper-parameter tuning

As described in the experiment section the main hyperparameters on which finetuning was conducted was the number of epochs, the batch size and the learning rate. No hyperparameter optimization framework like Optuna was used to find the optimal combination. The following search space was explored manually:
Epochs:2,3,4
Learning rate: 5e-5,3e-5,2e-5
Batch_size: 16,32

3.3 Optimization techniques

As optimization technique used in the experiments can be considered the linear warmup learning rate scheduling. Based on this scheduler the learning rate decreases linearly from the initial lr set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial lr set in the optimizer.

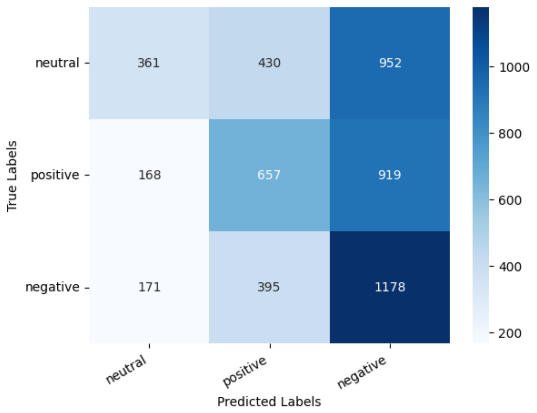


3.4 Evaluation

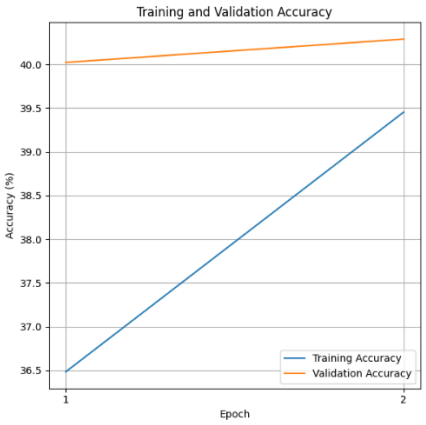
During evaluation the learning curves, classification reports and confusion matrices of all experiments for both finetuned models were plotted. For a more fruitful comparison it will be interesting to showcase the learning curves and confusion matrices of each experiment for each model and discuss on their differences.

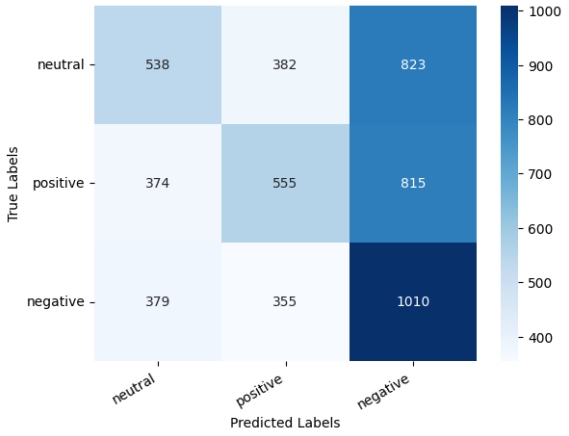
3.4.1 Learning Curves & Confusion Matrices.

GreekBERT



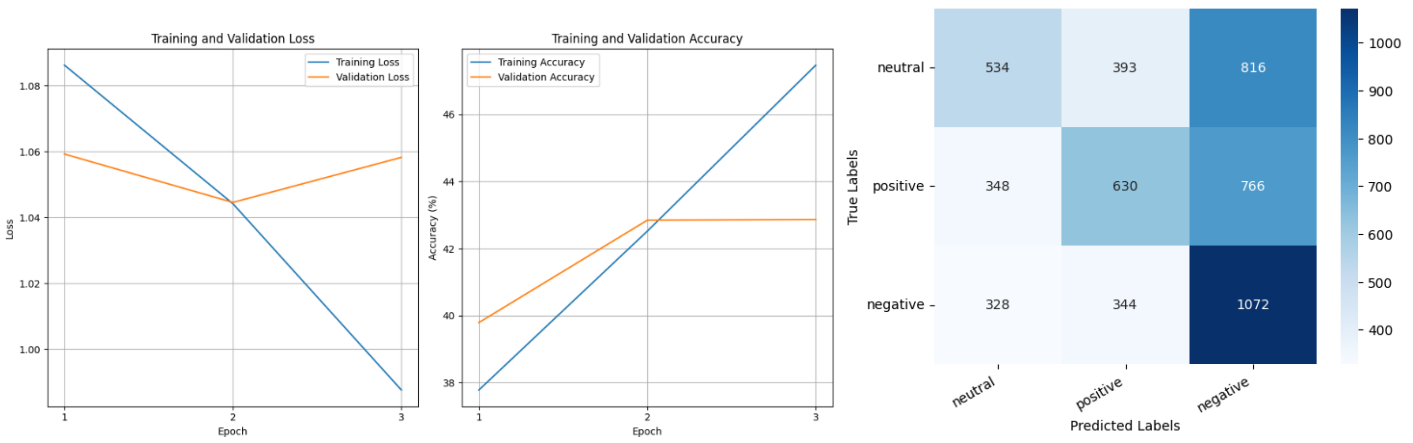
DistillGreekBERT



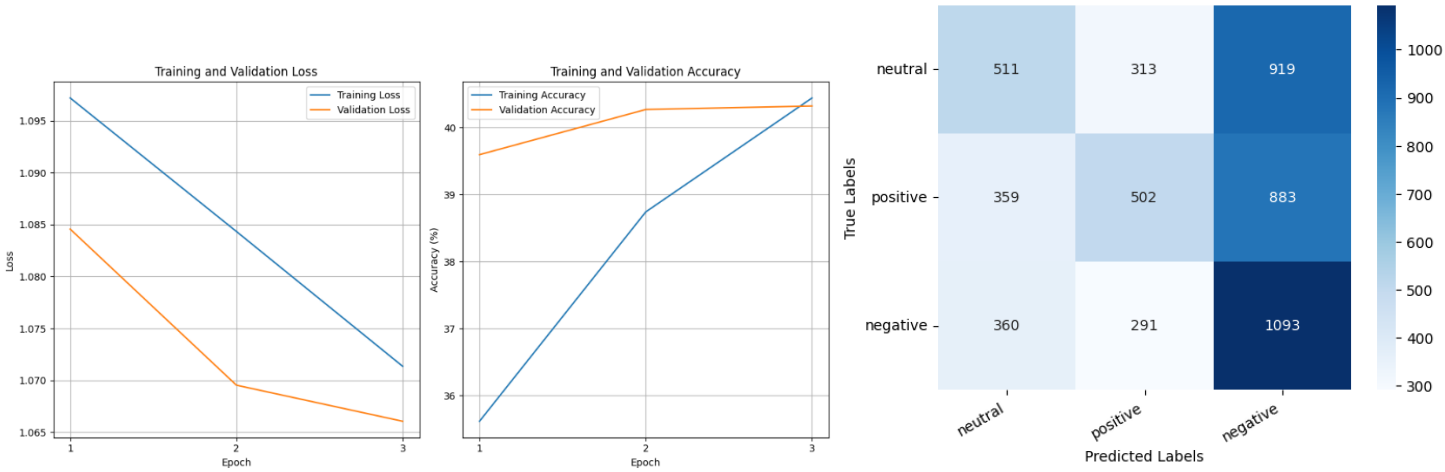


Experiment_1

GreekBERT



DistillGreekBERT



Experiment_2

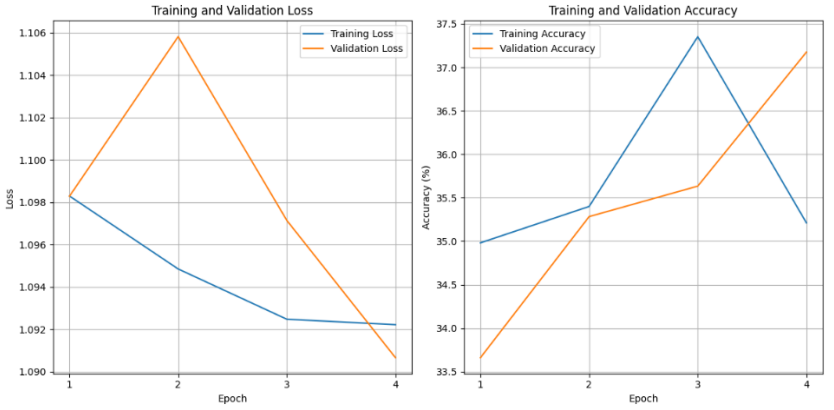
Experiment_1

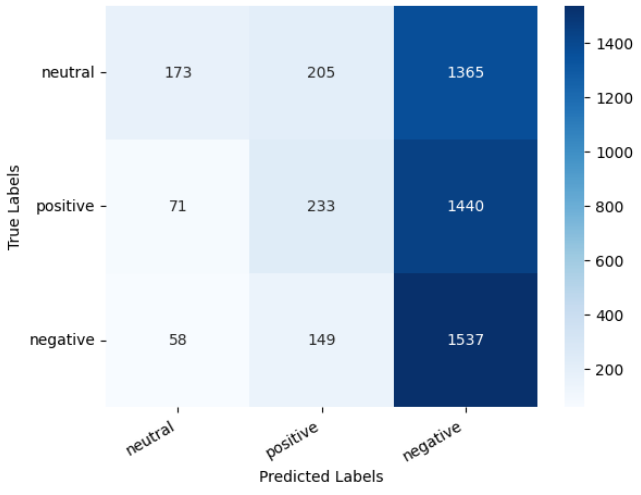
- DistillBERT model shows more underfitting, meaning that it needs more epochs and more complex model parameters to learn.
- We will see compared to the next experiments on the GreekBERT that the 2 epochs and batch size 32 will be the optimal for further experimentation on the learning rate value.

Experiment_2

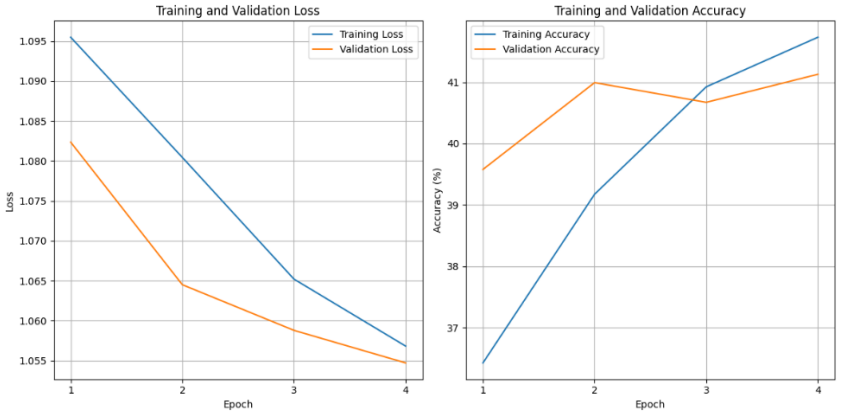
- For GreekBERT after 2 epochs training the model is overfitting on the train data
- For DistilGreekBERT the increase in epochs seem to help the model learn more about the data and not overfit.

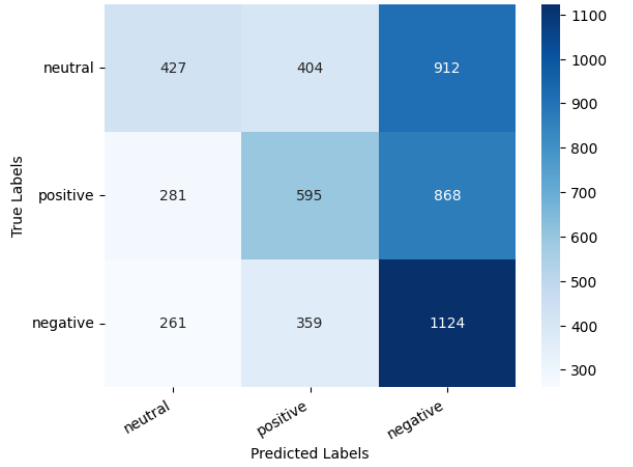
GreekBERT





DistillGreekBERT



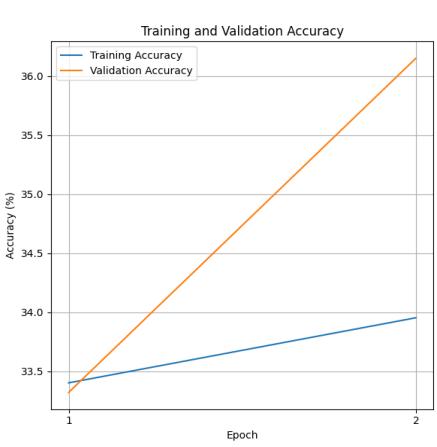


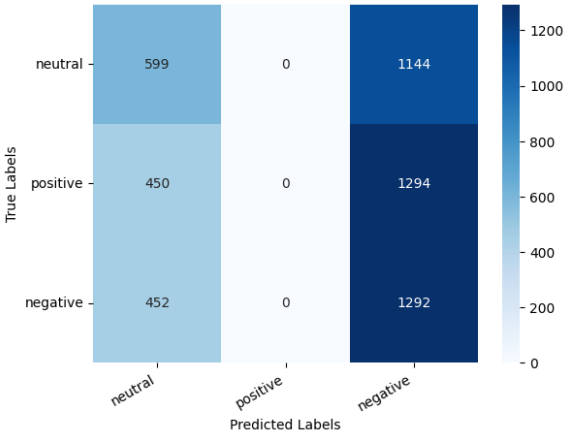
Experiment_3

Experiment_3

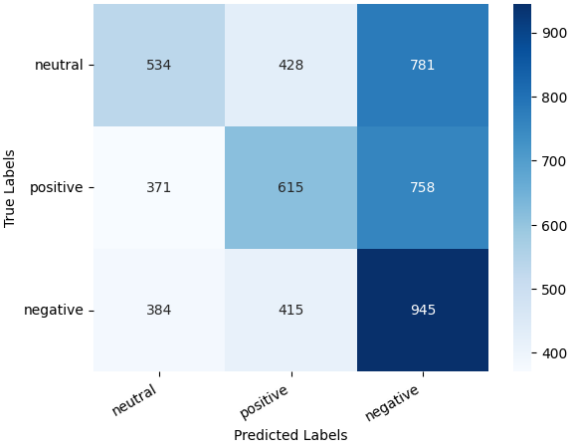
- For DistillGreek it seems that in 4 epochs the loss keeps decreasing, and the accuracy has a stable increasing tendency.
- The behavior of GreekBERT in validation loss and training accuracy cannot be explained by the given hyperparameters (batch size: 32, epochs, 4,lr:5e-5)

GreekBERT





DistillGreekBERT

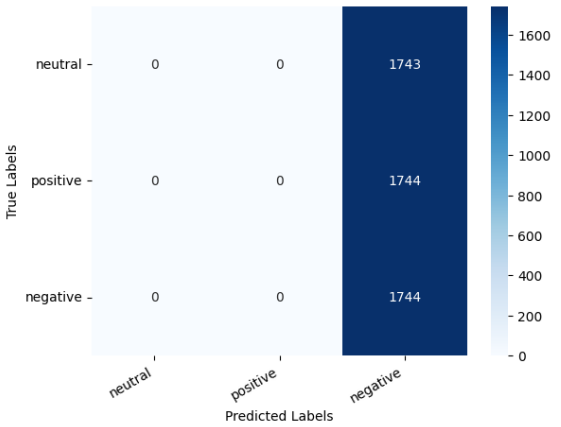


Experiment_4

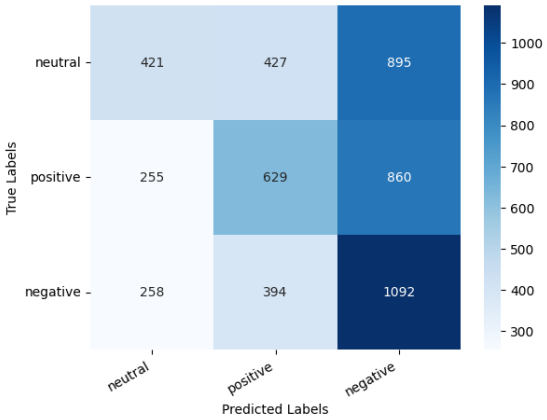
Experiment_4

- With a smaller batch size of 16 both models seem to be underfitting in the two epochs, while the DistillGreek model accuracy, shows potential improvement when adding more epochs.

GreekBERT



DistillGreekBERT

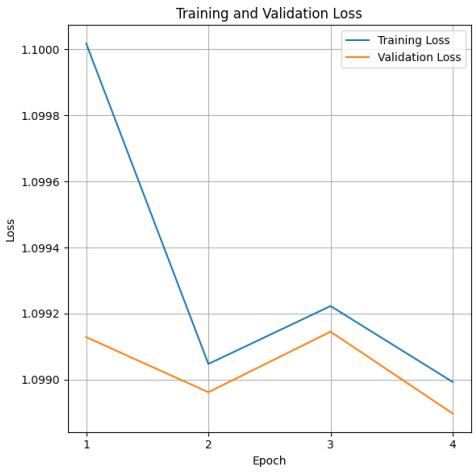


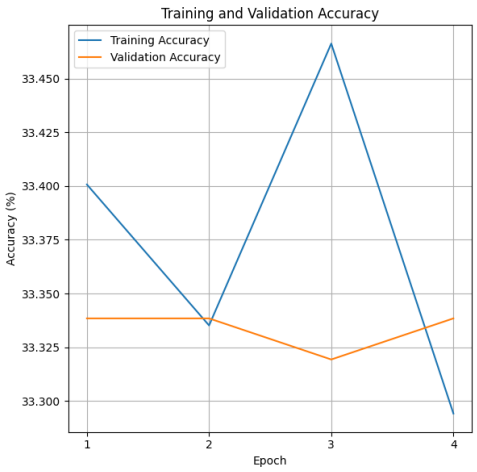
Experiment_5

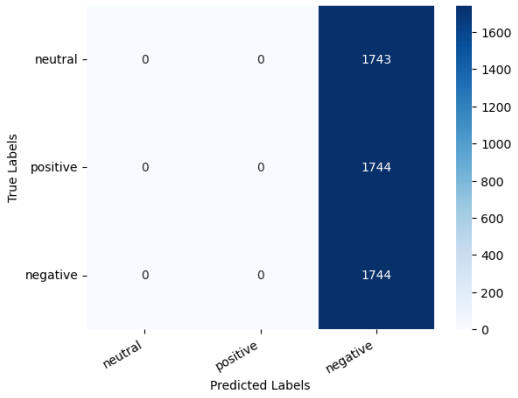
Experiment_5

- For GreekBERT model it seems that 2 epochs are the optimal for the model to not get confused and start making wrong predictions.
- The DistilGreek model shows that adding more training epochs helps the model converge.

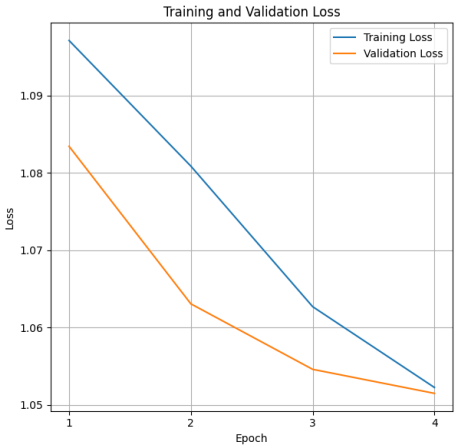
GreekBERT

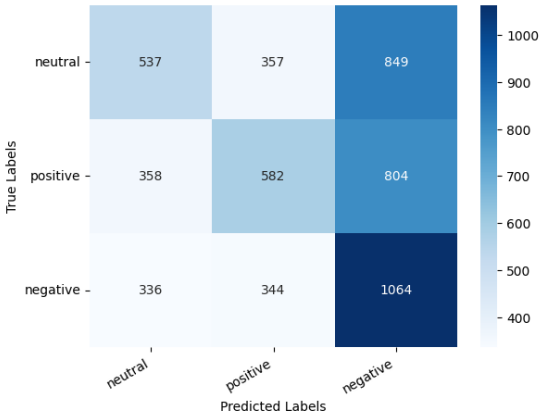






DistillGreekBERT



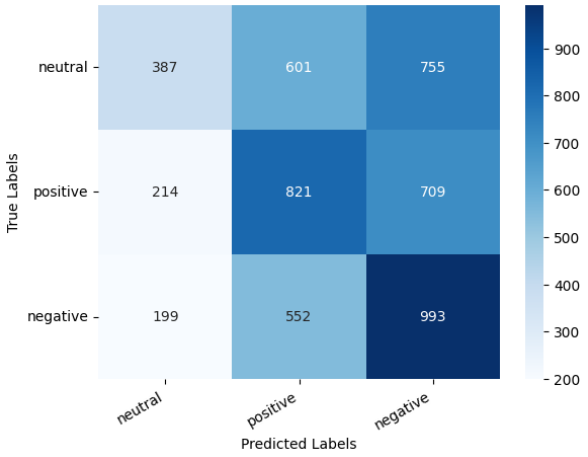


Experiment_6

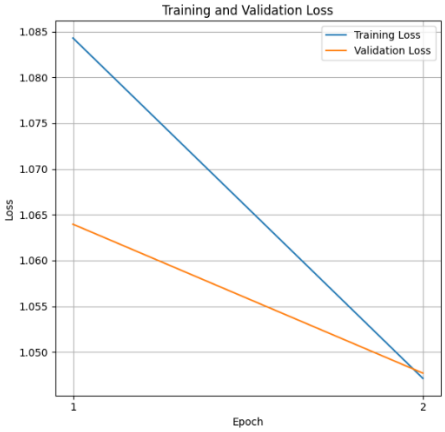
Experiment_6

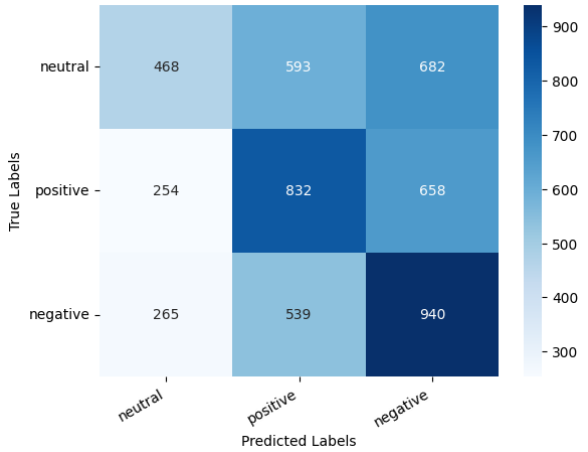
- For GreekBERT, even on batch size 16, we see that 2 epochs are the optimal, so that the model does not get confused.
- For DistilGreek 4 epochs and a batch size of 16 are the optimal values for the next experiments (7 and 8) about the learning rate value.

GreekBERT Experiment 7



GreekBERT Experiment 8

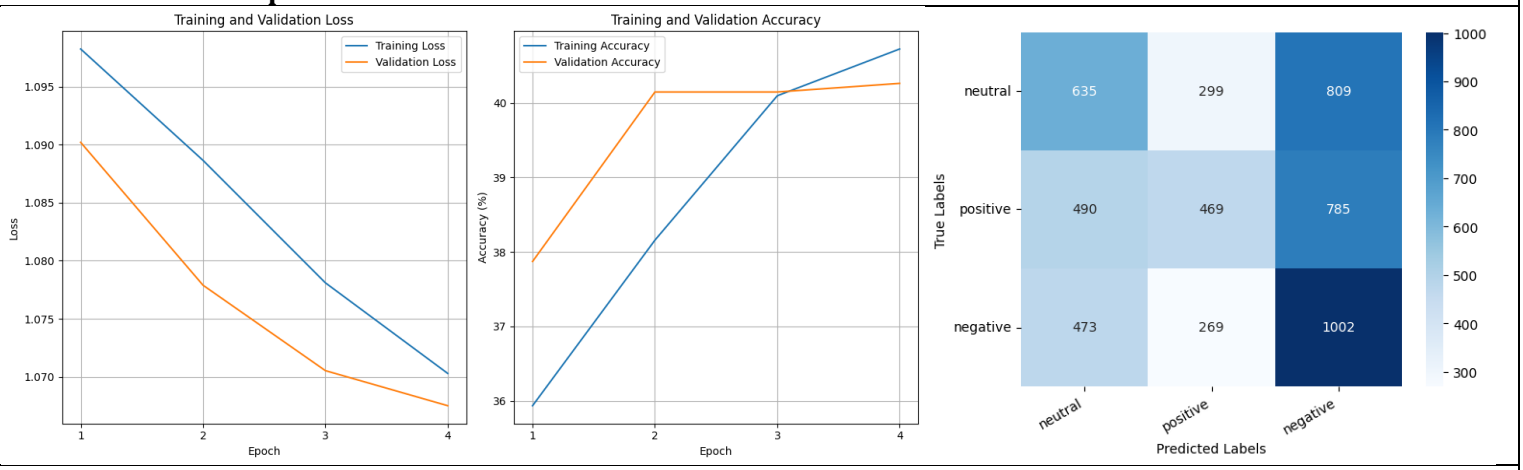




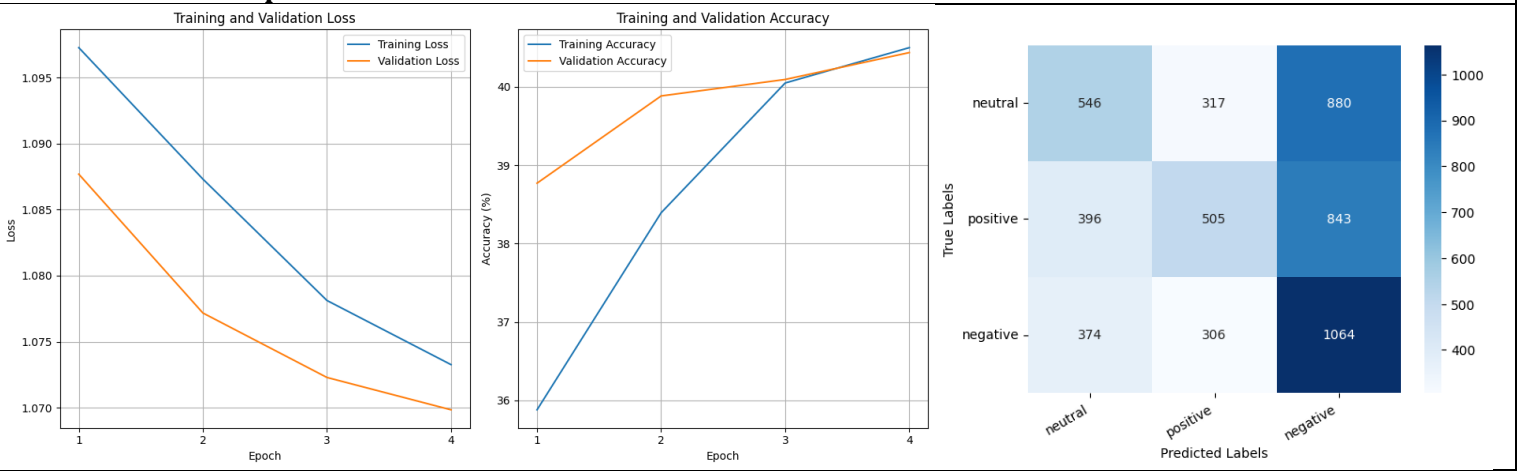
For GreekBERT (epochs_2, batch size:32)

- Both learing rates (3e-5 and 2e-5) help the model converge. A smaller learning rate helps the model achieve a slightly higher validation accuracy, while the loss shows potential for decreasing more that 1.045

DistilGreekBERT Experiment 7



DistilGreekBERT Experiment 8



For DistilGreekBERT (epochs_4, batch size:16)

- A smaller learning rate of 2e-5 helps the model converge and reach a slightly higher validation accuracy, while loss seems stabilized around 1.07

<u>Comments on confusion matrices</u>

Starting from the **DistilGreekBERT** finetuned classifiers it is obvious that they are all biased towards the negative class. On average the majority of correct predictions for positive and neutral class are consistently lower than those of the negative class.

Concerning the **GreekBERT** classifiers we observe different behavior in specific experiments settings. Experiments 1,2 and 3 indicate the same bias towards the negative class as reported for DistilBERT, with the 3rd experiment showing significant lower correct predictions for positive and neutral class compared to 1st and 2nd experiment. In experiment 4 the classifier is still biased towards negative class, but it does not predict positive tweets at all, while in experiments 5 and 6 the classifier predicts only negative tweets. Experiments 7 and 8 indicate that there is still struggle in prediction of neutral tweets, but the prediction behavior is comparably more balanced across the three classes, with the last one being chosen as the most appropriate for submission in the Kaggle competition.

Below wee the performance of all experiments in terms of accuracy and weighted precision,recall and f1-score.

- **Accuracy** measures the proportion of correctly classified cases from the total number of objects in the dataset. In our balanced dataset, accuracy can serve as F1-micro.
- **Precision** for a given class in multi-class classification is the fraction of instances correctly classified as belonging to a specific class out of all instances the model predicted to belong to that class. It measures the model's ability to identify instances of a particular class correctly.
- **Recall** in multi-class classification is the fraction of instances in a class that the model correctly classified out of all instances in that class.
- **F1-score:the** Harmonic mean of precision and recall for a more balanced summarization of model performance.
- **With Weighted averaging** the output average accounts for the contribution of each class as weighted by the number of examples of that given class. So, it is helpful in both balanced and imbalanced datasets.

Trials	Accuracy	Precision (weighted avg)	Recall (weighted avg)	F1 (weighted avg)
Experiment_1	0.42	0.45	0.42	0.40
Experiment_2	**0.43**	**0.44**	**0.43**	**0.42**
Experiment_3	0.37	0.44	0.37	0.29
Experiment_4	0.36	0.25	0.36	0.28
Experiment_5	0.33	0.11	0.33	0.17
Experiment_6	0.33	0.11	0.33	0.17
Experiment_7	0.42	0.43	0.42	0.41
Experiment_8 (submission)	**0.43**	**0.44**	**0.43**	**0.42**

Table 1: Trials GreekBERT

Trials	Accuracy	Precision (weighted avg)	Recall (weighted avg)	F1 (weighted avg)
Experiment_1	0.40	0.41	0.40	0.39
Experiment_2	0.40	0.42	0.40	0.39
Experiment_3	0.41	0.42	0.41	0.39
Experiment_4	0.40	0.41	0.40	0.39
Experiment_5	0.41	0.42	0.41	0.39
Experiment_6	**0.42**	**0.43**	**0.42**	**0.41**
Experiment_7	0.40	0.41	0.40	0.39
Experiment_8	0.40	0.41	0.40	0.39

Table 2: Trials DistilGreekBERT

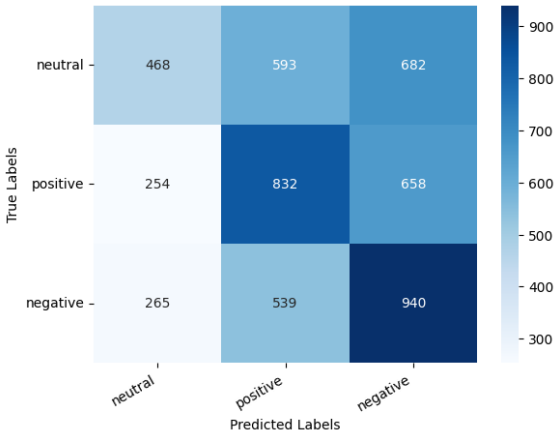
4 Results and Overall Analysis

4.1 Results Analysis

From the experimentation with GreekBERT and DistillGreekbert models the following observations were made:

- Both models showed comparable performance with different batch size, learning rate and epoch combinations.
- DistilGreekBERT reached a weighted F1 score of 0.41 with a batch size of 16, 4 epochs training and learning rate 5e-5.
- GreekBERT model was able to exceed the best results of previous projects reached a weighted F1 score of 0.42, with just 2 epochs training, a batch size of 32, and learning rate 2e-5
- Training time for an epoch with DistillBERT took on average 300 seconds(5minutes)/epoch, while with GreekBERT double time was needed.
- Considering all the calculated evaluation metrics reported we cannot say that the classifiers are accurately predicting all three classes. As reported above there is an overall bias towards the negative class.
- Considering the pretraining knowledge of GreekBERT on Greek text content it was expected that the finetuning approach would give results comparable to the previous projects.
- In this initial approach on finetuning GreekBERT for Sentiment Analysis task, it was attempted to keep the additional layer architecture as simple as possible by adding one additional Feedforward (hidden) layer, a ReLU activation function and an output layer. For future experimentation I could try adding dropout rate or adding more feedforward layers before the output layer to make a deeper network.
- Also, it was mentioned at the BertClassifier architecture that a hidden size of 50 neurons was used based on a tutorial from which I got inspired. So experimentation on the dimensionality of the additional hidden layer is recommended.
- Furthermore, experimentation with bigger batch sizes (e.g. 64,128) for GreekBERT and more epochs (e.g. 5,6) for DistilGreekBERT would be for further consideration.

4.1.1Best trial

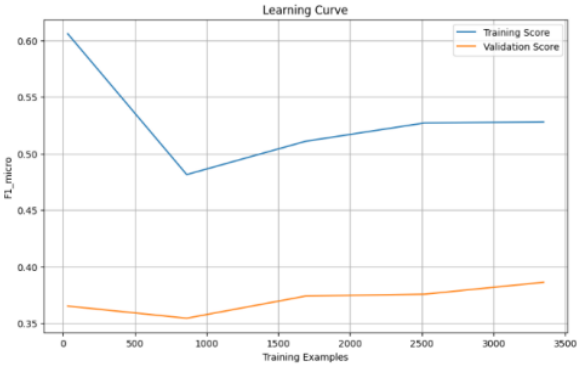


	precision	recall	f1-score	support
neutral	0.47	0.27	0.34	1743
positive	0.42	0.48	0.45	1744
negative	0.41	0.54	0.47	1744
accuracy			0.43	5231
macro avg	0.44	0.43	0.42	5231
weighted avg	0.44	0.43	0.42	5231

Experiment_8(Kaggle submission)
GreekBERT (batch_size:32, n_epochs:2,lr:2e-5)

4.2 Comparison with the first project

Comparing the best model results with those of the best model on the first project (TfIdf unigrams) max_features 1000 +3 numeric, L1 regularization, saga, C: 1.0, trained with Logistic Regression), it was expected to see an improvement as we did. It is worth noting that on the first project a different data partitioning approach was followed. Both train and validation data were splitted into train/test and dev/test with a ratio of 80/20. This is why on the classification report below we see less data (1047). Considering learning curves in project_1 we plotted learning curves of F1 score on train and validation data. It was observed that all models were overfitting on the train data, and this was accounted due to the use of a linear model which is probably not suitable for our data since they are not linearly separable.



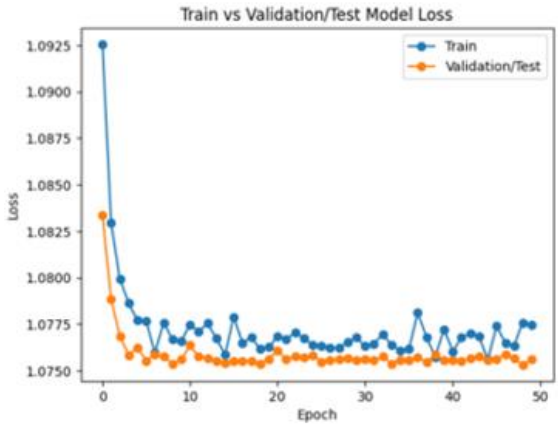
Classification report:

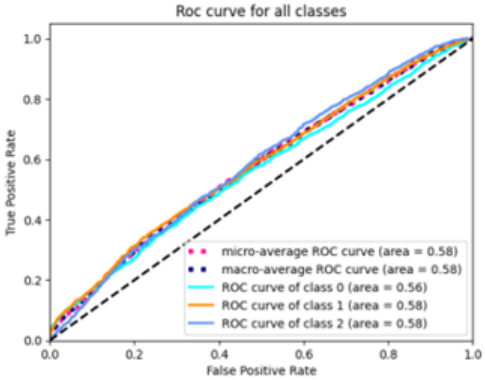
	precision	recall	f1-score	support
NEGATIVE	0.39	0.34	0.36	360
NEUTRAL	0.38	0.50	0.44	342
POSITIVE	0.43	0.34	0.38	345
accuracy			0.40	1047
macro avg	0.40	0.40	0.39	1047
weighted avg	0.40	0.40	0.39	1047

4.3 Comparison with the second project

There are plenty of differences between the 2nd and the 4th project, yet these differences do not seem to improve the final evaluation results considering the overall experimentation in finetuning GreekBERT and DistilGreekBERT. It is still worth pointing some of these differences out.

Project_2	Project_4
• 1 layer • Unidirectional FFNN • Word2vec (200d) • He(weight initialization) • Regularization techniques (scaling,dropout,batch normalization) • Randomized Leaky ReLU • RMSProp • Lr: 0.0004 • Scheduler: exponential • Epochs:50 • Batch_size: 128	• 12 Bert layers +1additional for this task • Bidirectional encoder • BERT(768d)+positional embeddings • Pretrained weights for parameter initialisation • Regularization techniques (dropout,layer normalization, Residual connections) • GELU • Adam • Lr: 0.00002 • Scheduler: linear warmup • Epochs:2 • Batch_size: 32



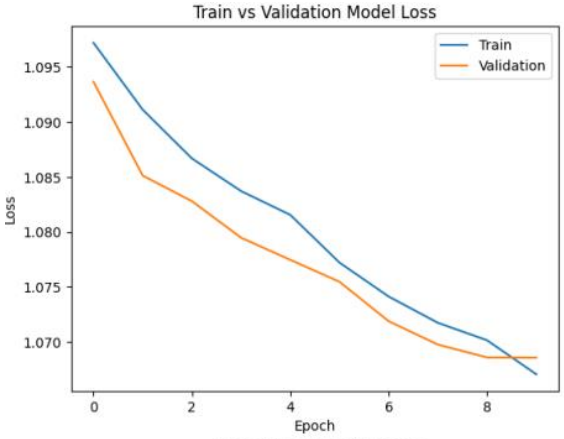


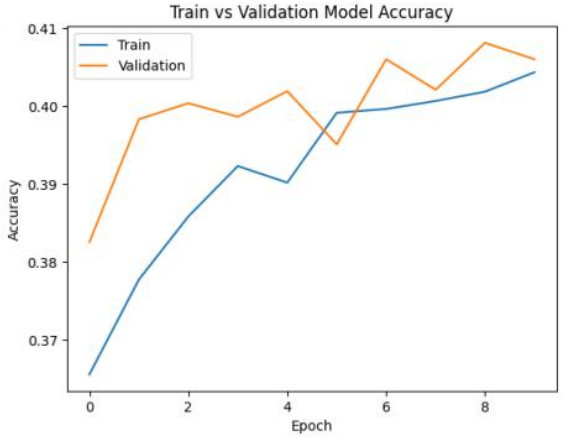
	precision	recall	f1-score	support
0	0.41	0.29	0.34	1742
1	0.41	0.38	0.40	1744
2	0.39	0.52	0.45	1744
accuracy			0.40	5230
macro avg	0.40	0.40	0.39	5230
weighted avg	0.40	0.40	0.39	5230

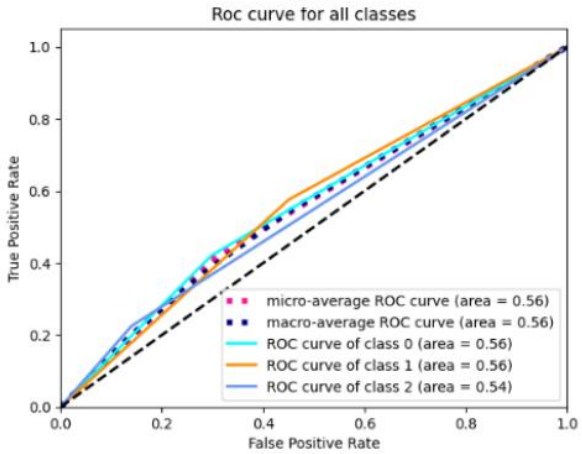
4.4 Comparison with the third project

Project_3	Project_4
• 2 layers • Bidirectional LSTM • Word2vec (200d) • ReLU • Lr: 0.0001 • Epochs:10 • Batch_size: 256 • Gradient_clipping_thres: 5.0 • Dot-product attention	• 12 Bert layers +1additional for this task • Bidirectional encoder • BERT(768d)+positional embeddings • GELU • Lr: 0.00002 • Epochs:2 • Batch_size: 32 • Gradient_clipping_thres: 1.0 • Bidirectional Self-attention with 12 attention heads

Compared to the simple dot-product attention applied to the bi-lstm in the 3rd project the bi-directional multiheaded self-attention not only does it help the current token attend to its context bidirectionally, but also through the attention heads that are trained to represent a different functionality, they can help achieving a deeper representation of the current token.







	precision	recall	f1-score	support
0	0.41	0.42	0.42	1744
1	0.39	0.58	0.47	1744
2	0.44	0.23	0.30	1742
accuracy			0.41	5230
macro avg	0.42	0.41	0.39	5230
weighted avg	0.42	0.41	0.39	5230

Considering the differences in the projects above and their evaluation results we acknowledge again the power of the pretrained Bidirectional Encoder only model, named BERT, which managed with only 20 minutes of finetuning (10 minutes /epoch), a batch size of 32 and a learning rate of 2e-5 to outperform all previous projects. We also acknowledge the computational efficiency and performance of models trained with Knowledge Distillation, which managed to give results comparable to all previous projects, including the best performing GreekBERT model. Nevertheless, considering the specific data on which the experiments were conducted, it is needless to repeat that we are working with a dataset with a lot of false positives, false negatives and false neutrals. One step towards improving this project is definitely data curation and more in-depth exploration of the tweet content to understand deeper patterns indicating emotion and

specific handling of tweets with no textual content, containing only links and hashtags. We understand and appreciate the originality of this real-life dataset, but it was not a fruitful source for proper experimentation and inspection of the applicability of our ideas on improvements, since as it was observed no project managed to reach a score higher that 40-43%, which for a multiclass classification task is equal to random guessing.

5 Bibliography

References

[1] *BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick*. (2019, July 22). Mccormickml.com. https://mccormickml.com/2019/07/22/BERT-fine-tuning/

[2] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://arxiv.org/pdf/1810.04805.pdf

[3] *How to interpret a confusion matrix for a machine learning model*. (n.d.). Www.evidentlyai.com. https://www.evidentlyai.com/classification-metrics/confusion-matrix

[4] Karavangeli, E. (2023). *DistilGREEK-BERT: A distilled version of the GREEK-BERT model*. Retrieved March 8, 2024, from https://pergamos.lib.uoa.gr/uoa/dl/object/3338746/file.pdf

[5]nlpaueb/bert-base-greek-uncased-v1 · Hugging Face. (n.d.). Huggingface.co. https://huggingface.co/nlpaueb/bert-base-greek-uncased-v1

[6] Tran, C. (2020, April 15). *Tutorial: Fine-tuning BERT for Sentiment Analysis*. Skim AI. https://skimai.com/fine-tuning-bert-for-sentiment-analysis/

[7]*ValueError: TextEncodeInput must be Union[TextInputSequence, Tuple[InputSequence, InputSequence]] - Tokenizing BERT / Distilbert Error*. (n.d.). Stack Overflow. Retrieved March 6, 2024, from https://stackoverflow.com/questions/63517293/valueerror-textencodeinput-must-be-uniontextinputsequence-tupleinputsequence