UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Antigoni Stefanou*
sdi: sdi2000281

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Spring Semester 2025*

Contents

1. Abstract

In this assignment, the task is to categorize tweets, depending on whether the sentiment behind them is positive or negative. For this purpose, the pre-trained models BERT and DistilBERT will be finetuned with the help of the Optuna framework. The relevant notebooks can be found here:
https://www.kaggle.com/code/antigonistefanou/sdi2000281-hw3-bert
https://www.kaggle.com/code/antigonistefanou/sdi2000281-hw3-distilbert

2. Data processing and Analysis

2.1. Pre-processing

The first task in this assignment is the pre-processing, the goal of which is to remove the noise often found in tweets, in order to improve the model's performance. The following methods will be used to clean the datasets for both models.

1. Remove all HTML character entities since BERT does not have a way to convert them to conventional punctuation marks and special characters. These entities start with an ampersand and end with a semicolon (ex. <), so they can be found using RegEx.

2. Remove non-ASCII characters. The BERT model that was used has been trained on English text so it will not be able to recognize non-latin alphabets.

When training a sentiment classifier, it is common practice to employ additional methods, such as removing capitalization, shortening elongated words, removing stopwords or applying lemmatization. However, BERT and DistilBERT have been trained on raw text and perform better on lightly processed data. In fact, proper tuning of the model is much more important than excessive pre-processing when using BERT embeddings.[6]

2.2. Tokenization and Encoding

After the data is cleaned, it needs to be properly prepared for the model.

The first step (tokenization) was handled by the Hugging Face pre-trained tokenizers BertTokenizer and DistilBertTokenizer, using the bert-base-uncased and distilbert-base-uncased vocabulary files respectively. Along with tokenizing the data, they are responsible for the encoding and managing special tokens, such as the masks.[3]

Following the tokenization, a Custom Dataset Class is defined, such that accepts sequences of varying lengths -something a regular Tensor dataset cannot do- and is compatible with established Hugging Face methods.

```
class CustomDataset(Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.
   encodings.items()}
```

```
        if self.labels is not None:
            item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])
```

Listing 1: Custom Dataset Class

The training, validation and testing custom datasets are then converted to Dataloaders, with the help of a Data Collator that will form the batches. While the PAD tokens (added by the tokenizer) do not affect the model's performance, they impact the computational speed.[8] This problem is combated with the use of DataCollatorWithPadding, a collator that dynamically pads to the longest sequence in each batch.

2.3. Analysis

After finishing the pre-processing step, the shortest tweet overall is 1 word, while the longest is 57 words. The distribution of tweet lengths for all datasets is shown below.

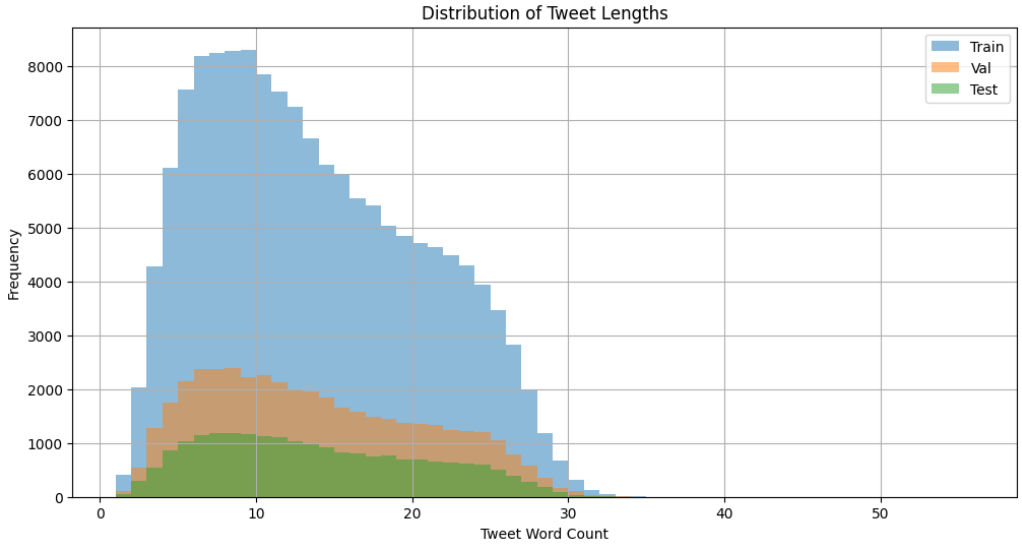


Figure 1: Distribution of Tweet Lengths after Pre-processing

Subsequently to the padding process, the shortest sequence is 35 tokens and the longest is 118 tokens. Below is the distribution of tweet lengths for all datasets.

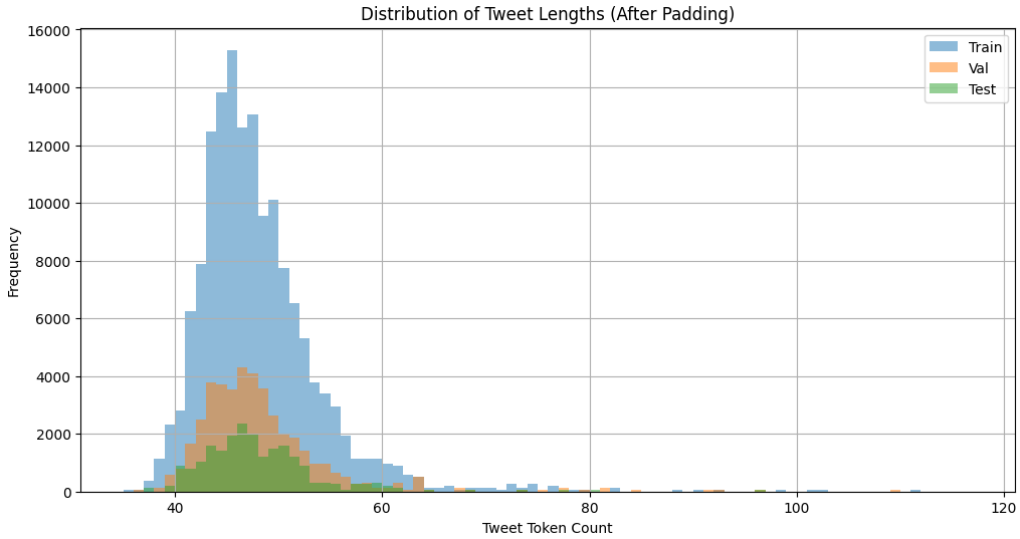


Figure 2: Distribution of Tweet Lengths after Padding

While most tweets were between 1 and 20 words, the vast majority of sequences are now between 40 and 60 tokens. This fact highlights the need for dynamic padding even more. The tokenization process and the addition of special tokens can greatly lengthen each tweet, with some reaching over 100 tokens. Therefore, in order to speed up the training process, it is important to pad each batch the minimum amount.

3. Algorithms and Experiments

3.1. Algorithms

First and foremost, the functions to be used in the training and evaluation of the models must be defined. The training function includes

- a progress update that is printed every 600 batches
- a standard forward pass, that calculates the predictions
- a standard backpropagation step, during which the gradients are calculated

The evaluation function predicts the validation dataset's labels using the trained model and then calculates its loss and accuracy score. The framework used for the above routines is Torch's Python implementation, PyTorch[10].

In order to better visualize the model's performance, two more functions are implemented: "print_plot_metrics" and "extraction". The "print_plot_metrics" function calculates the accuracy, F1 and ROC AUC scores, the confusion matrix and the distribution of probabilities and then plots them. The "extraction" function collects the CLS tokens and labels from a given dataloader, as well as the calculated probabilities and returns them. This method is used later when plotting UMAPs.
The aforementioned metrics are thoroughly explained in section 3.2.

3.2. Evaluation

The following metrics will be used to evaluate the models.

3.2.1. Accuracy.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy is a metric that measures how often a model correctly predicts the outcome, in this case the tweet's sentiment. A high accuracy score means that the model can correctly categorize the data. However, it is not enough on it's own, as a very high accuracy is also a byproduct of overfitting.

3.2.2. Per-Class Metrics Barplot. A barplot that showcases the accuracy, precision, recall and F1 score for each of the two classes. With it, it becomes apparent if the model is biased towards one of the classes.

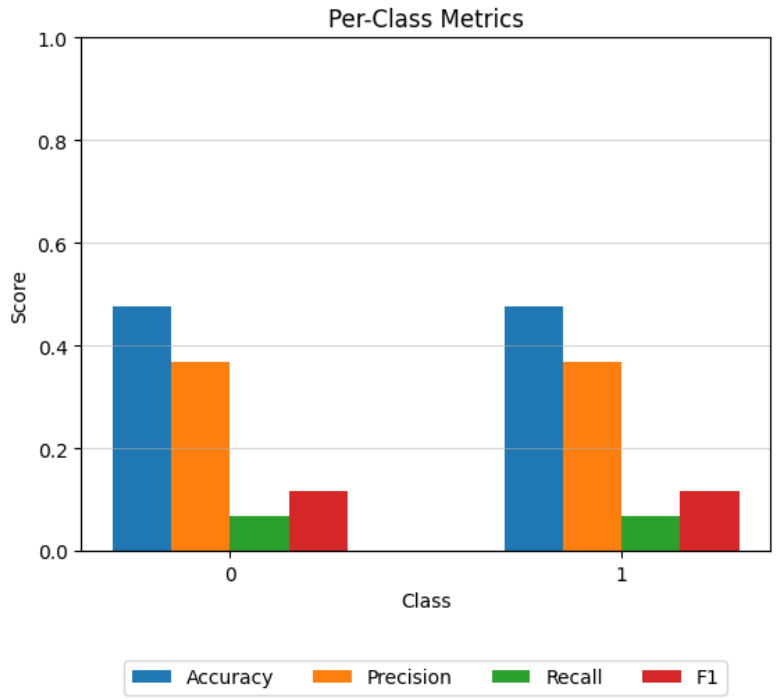


Figure 3: DistilBERT - Base Case Per-Class Metrics Barplot

3.2.3. Confusion matrix. The confusion matrix is a 2x2 table with the number of correct predictions (true positives and true negatives) and incorrect predictions (false positives and false negatives) for both classes. It is a very important metric, since it clearly presents the model's weak points.

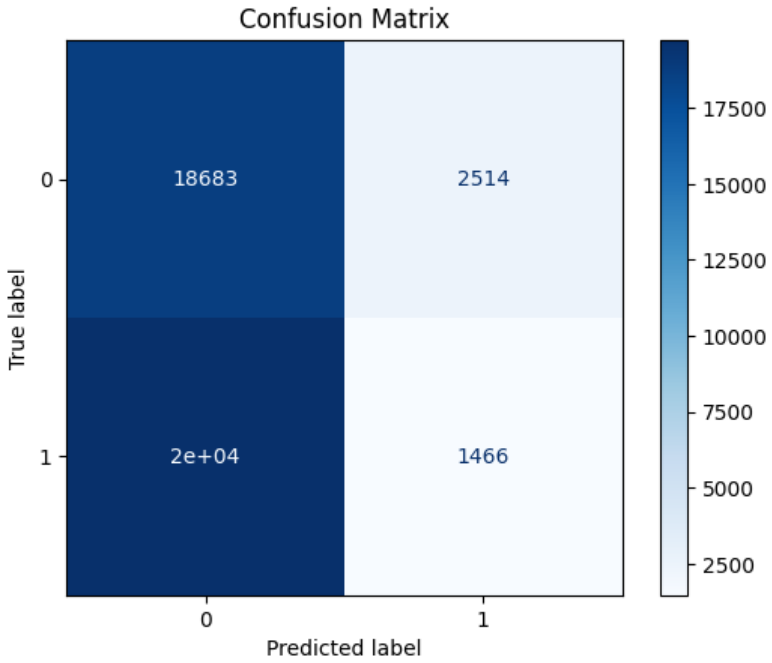


Figure 4: DistilBERT - Base Case Confusion Matrix

3.2.4. ROC Curve. The ROC curve shows the performance of a binary classifier. It is the plot of the true positive rate (TPR) against the false positive rate (FPR). The area under the ROC curve (AUC) is a good way to summarize in a single number the diagnostic ability of the classifier.[1] The more expansive the area, the better the classifier.

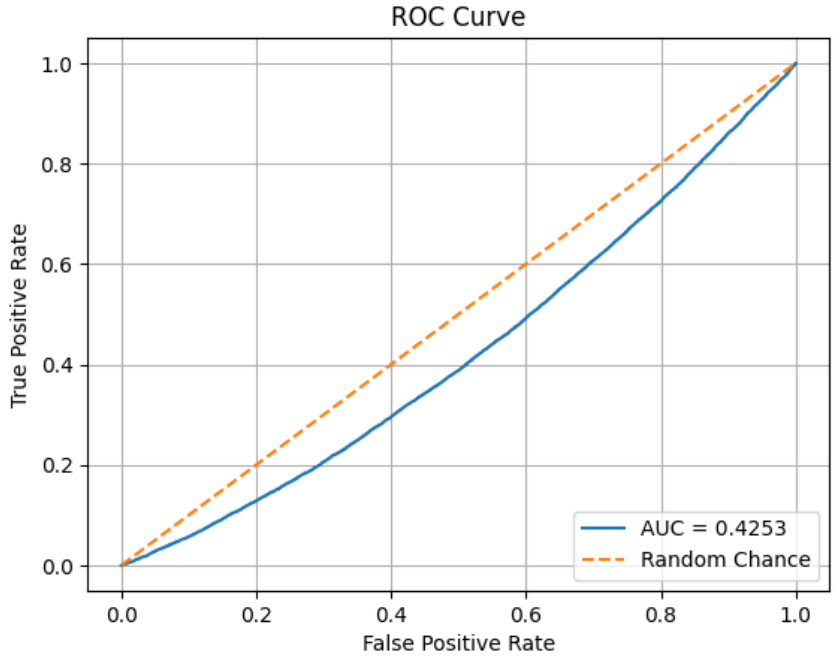


Figure 5: DistilBERT - Base Case ROC Curve

3.2.5. Distribution of Probabilities. This figure shows the probabilities generated by the model before they are rounded to 0/1. Therefore, it presents well how certain the model is with its predictions in general, as well as its confidence for each class.

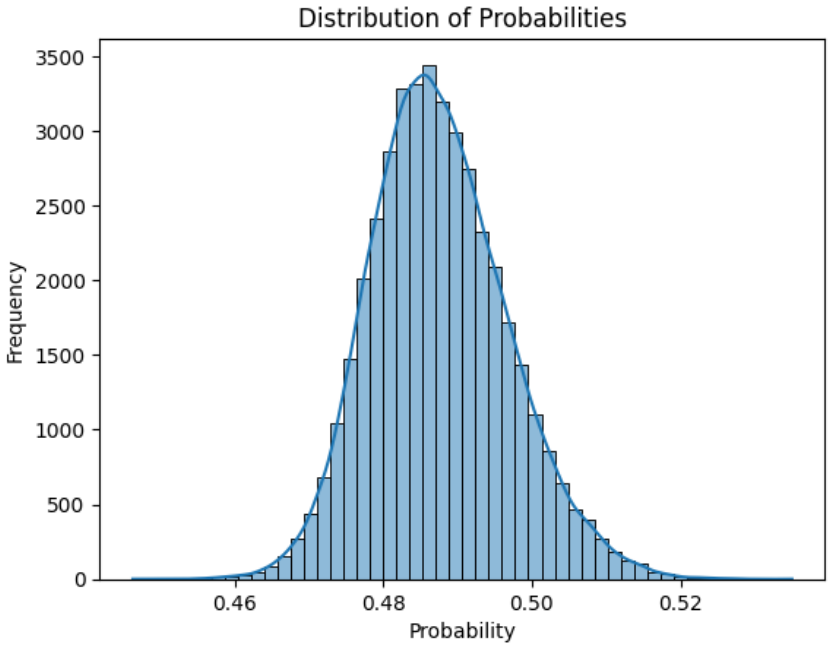


Figure 6: DistilBERT - Base Case Distribution of Probabilities

3.2.6. UMAP. Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization, but also for general non-linear dimension reduction. It greatly preserves the data's global structure while also having a superior run time performance.[9] This is achieved by constructing a weighted graph (in this case, in a 2-dimensional space) that resembles the initial input and then repeatedly perturbing its points until it's as close to the original as possible. When coloured by label, the plot shows how well the model can separate the classes. When coloured by probability, it demonstrates the model's certainty in its predictions.

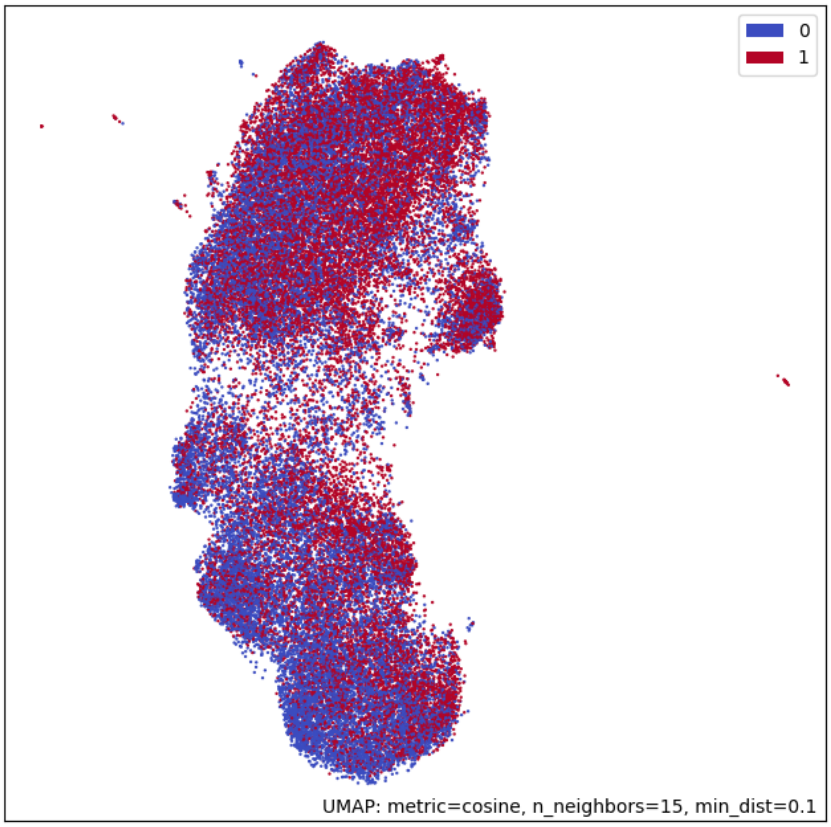


Figure 7: DistilBERT - Base Case UMAP Coloured by Label

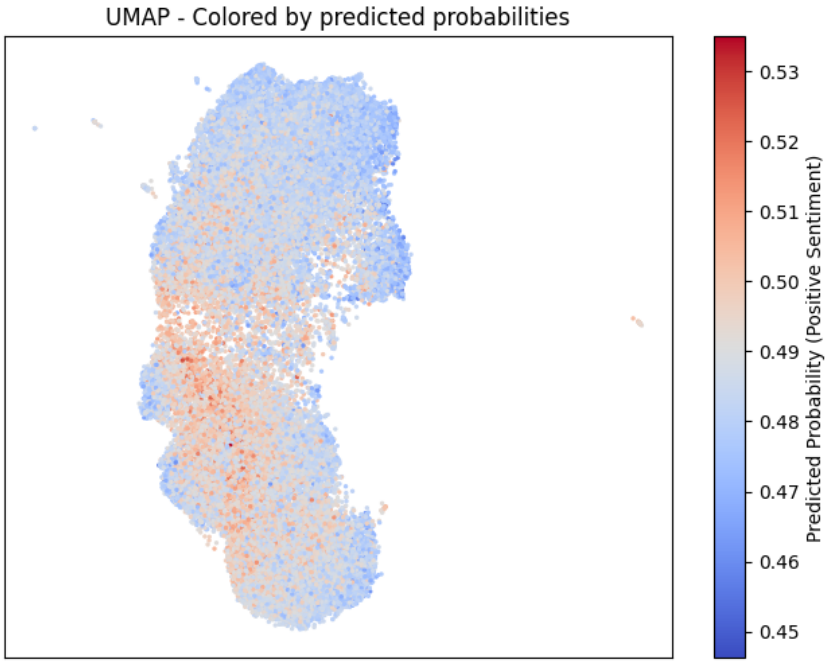
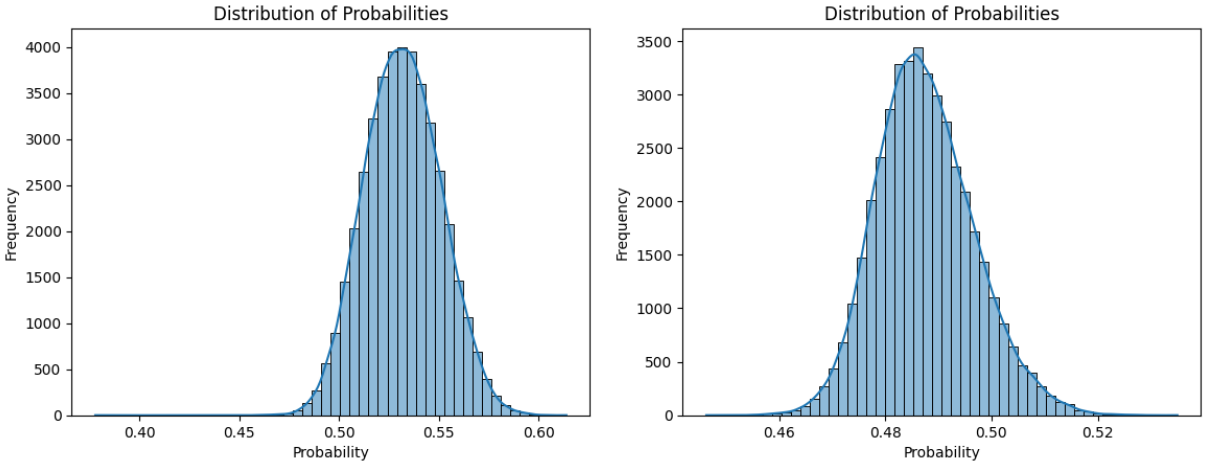


Figure 8: DistilBERT - Base Case UMAP Coloured by Probability

3.3. Experiments

The experimentation process was the same for both models. Initially, a baseline was established in order to effectively tackle this problem. More specifically, a pretrained model for sequence classification was called (BertForSequenceClassification.from_pretrained and DistilBertForSequenceClassification.from_pretrained) and evaluated. The DistilBERT model yielded a validation accuracy of 0.48 while the BERT model reached a score of 0.51 - the scores that are to be improved upon. The distribution of probabilities for each model is showcased below.



(a) BERT (b) DistilBERT

Figure 9: Distribution of probabilities

Without any finetuning the models are highly uncertain in their predictions and seem to be deciding each tweet's class randomly, with the BERT model leaning slightly towards the positive class.

Since each model's architecture is already established, the next step is tuning the hyperparameters. There are several choices to consider, namely the optimizer, the learning rate, the scheduler and it's warm up steps, as well as the number of training epochs. Since BERT and DistilBERT models need only a few epochs of training, the epoch number was chosen manually through trial and error. The rest of the hyperparameters' values were determined using the Optuna optimization framework, which is analyzed in section 3.4. All the hyperparameters, with the exception of the learning rate, were then initialized accordingly and another Optuna study was run, in order to find the most optimal learning rate. This decision stems from the fact that when working with models like BERT and DistilBERT, tuning all the optimizer's hyperparameters makes no substantial difference in performance. Choosing a well performing optimizer and then only tuning the learning rate is as good as tuning all the hyperparameters. [4]

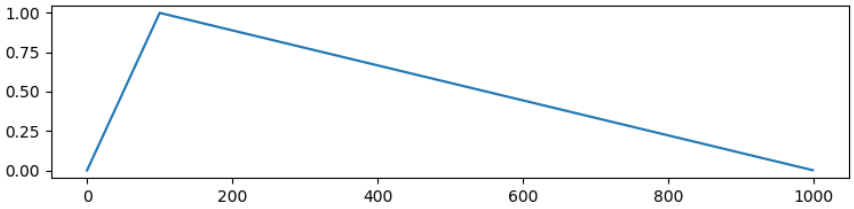
Further experimentation came with the merging of the training and validation datasets. This approach strengthened the model's generalization abilities, as it provided a larger data pool. Thus, any potential issues with underfitting were overcome and the accuracy score of both models was raised on the testing dataset.

3.4. Hyper-parameter tuning and Optimization techniques

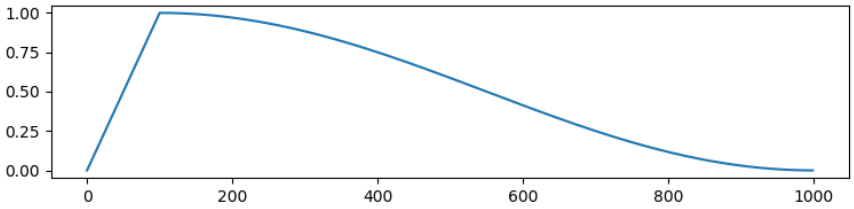
In order to achieve optimal results, the models' hyperparameters need to be tuned. This was achieved with the help of the Optuna optimization framework.

For the DistilBERT model two studies were created. The first study suggested the optimal values for the following parameters through 30 trials.

- optimizer, choosing between Adam and AdamW. Adam is a broadly used optimizer, while also being straightforward to implement, computationally efficient, with little memory requirements. [5] AdamW is a version of Adam that applies weight decay directly during the parameter update, leading to substantial improvement in Adam's generalization performance. [7]
- initial learning rate, from the range [1e-5, 5e-3]
- scheduler, choosing between linear with warmup and cosine with warmup. Both schedules decrease the learning rate from the initial value set in the optimizer to 0, following the values of the corresponding function. This happens after a warmup period, during which the learning rate increases linearly between 0 and the initial value set in the optimizer. [2] The process described is visualized in Figure 10.
- warm up steps for the scheduler, from the range [0, 750]



(a) Linear schedule with warmup



(b) Cosine schedule with warmup

Figure 10: Learning Rate Schedules, from the Hugging Face Documentation

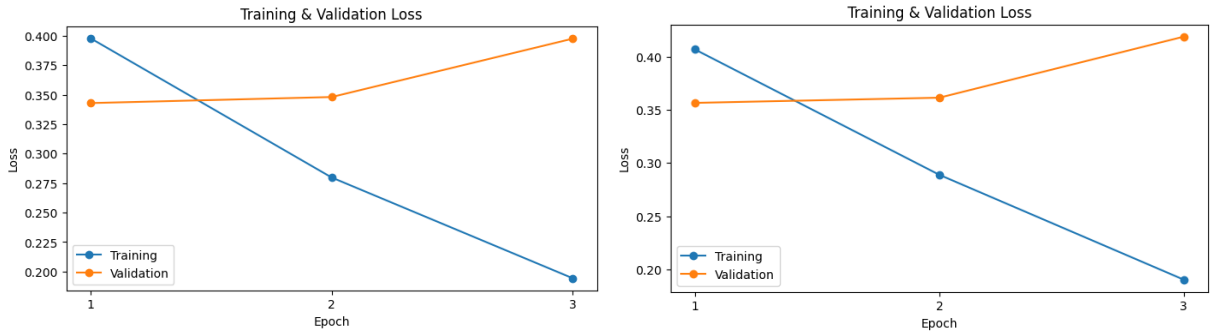
After these values were determined, the next goal was finding the optimal learning rate. For this reason, a second study was called, testing learning rate values in the range [1e-5, 5e-3] over 30 trials.

For the BERT model, one study was created that tested learning rate values in the range [1e-5, 5e-3] across 5 trials, while the rest of the hyperparameters were kept constant. They were initialized according to the results of the first DistilBERT study.

4. Results and Overall Analysis

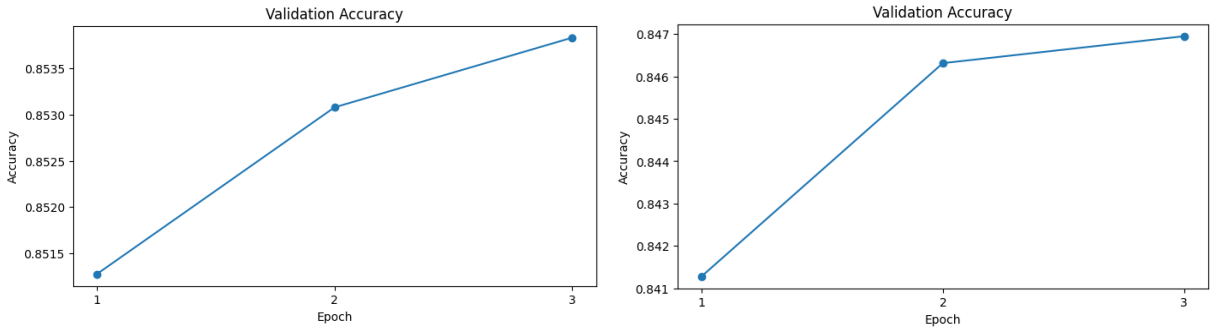
4.1. Results Analysis

During training, the following loss and accuracy scores were observed. Both models start overfitting after the second epoch, as indicated by the incline in validation loss. However, because the validation accuracy also increased -especially for the BERT model-, the training was allowed to continue for one more epoch. This way, the models would reap the benefits of a longer training period, without damaging their generalization capabilities.



(a) BERT (b) DistilBERT

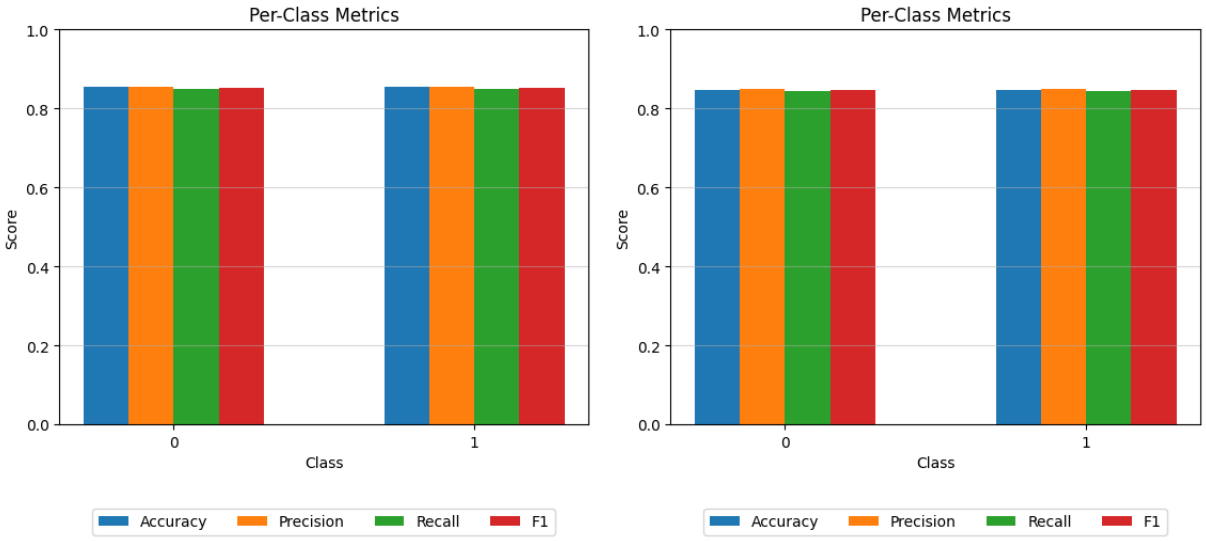
Figure 11: Training & Validation Loss



(a) BERT (b) DistilBERT

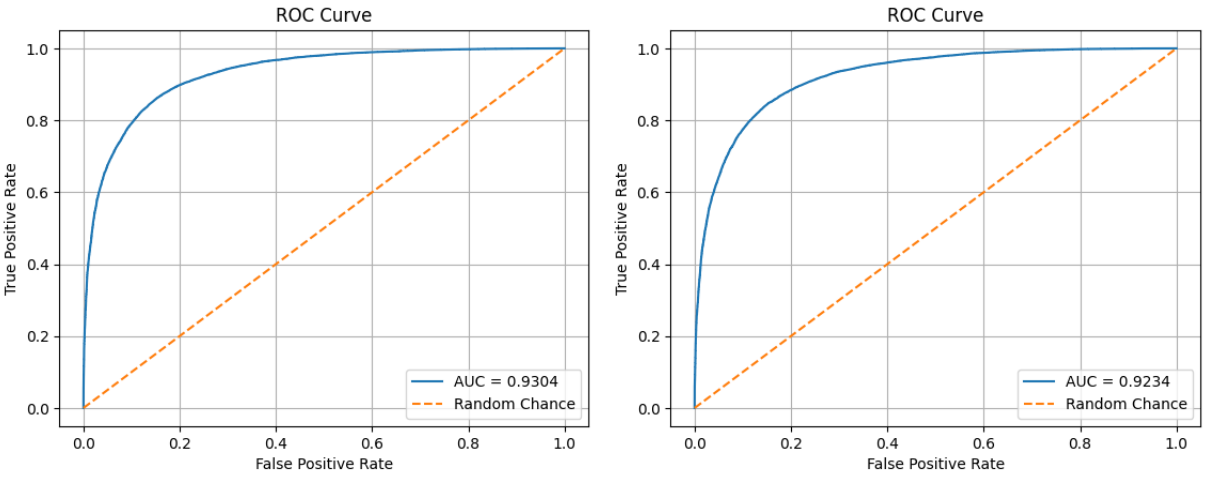
Figure 12: Validation Accuracy

After the finetuning and training process, the models can correctly predict a tweet's sentiment relatively well, with both DistilBERT and BERT reaching an accuracy score of 0.85 on the validation dataset. The precision, recall, F1 and ROC AUC scores are also similarly high (as seen in Figures 13 and 14).



(a) BERT (b) DistilBERT

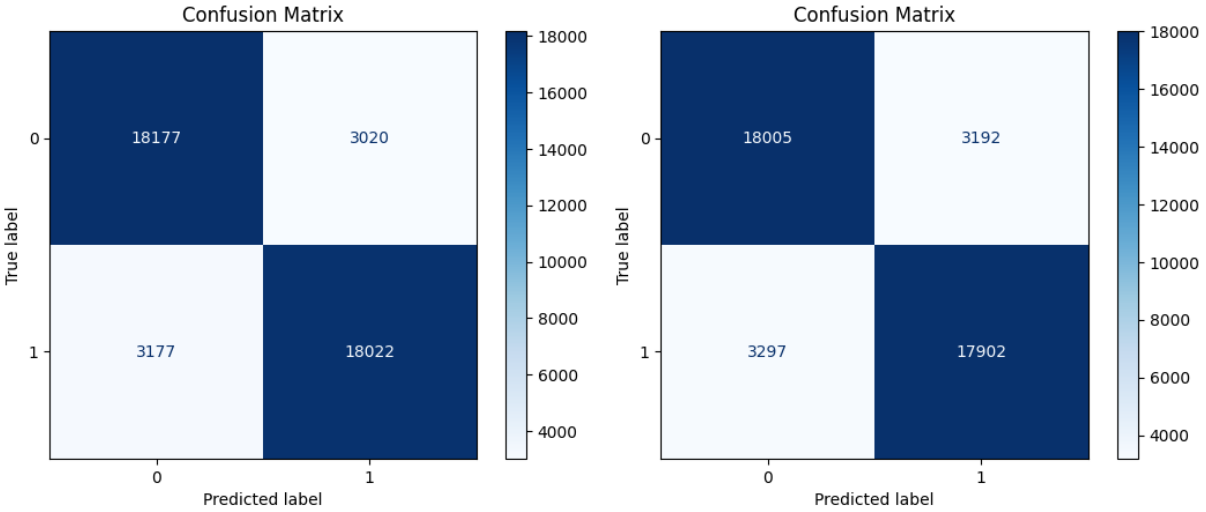
Figure 13: Per-Class Metrics Barplot



(a) BERT (b) DistilBERT

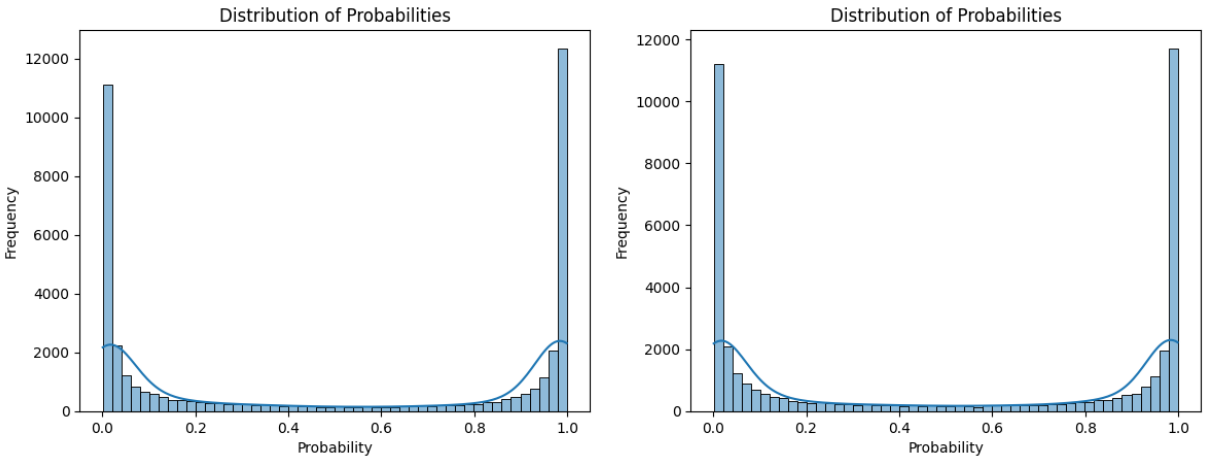
Figure 14: ROC Curve

The confusion matrix (Figure 15) shows that the models are able to correctly ascertain each tweet's class, with very few false positives and false negatives. Coupled with the probability distribution (Figure 16), which shows the majority of predictions concentrated around the decision boundaries, it becomes apparent that both models are robust. It could be argued that, because some predictions are near intermediate values, further development is needed. This is where the nature of the data is of particular importance. Human expression, and by extension Tweets, is not always inherently positive or negative in sentiment. Ergo, any observed uncertainty might be a reflection of the nuance and ambiguity of language, rather than the model's limitations.



(a) BERT (b) DistilBERT

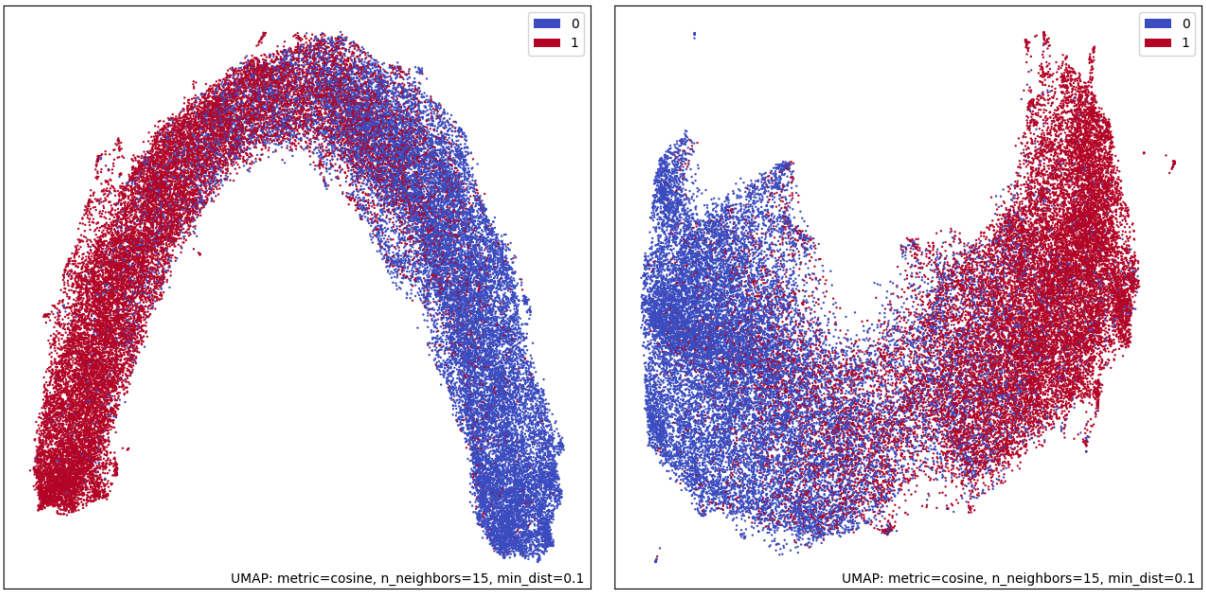
Figure 15: Confusion Matrix



(a) BERT (b) DistilBERT

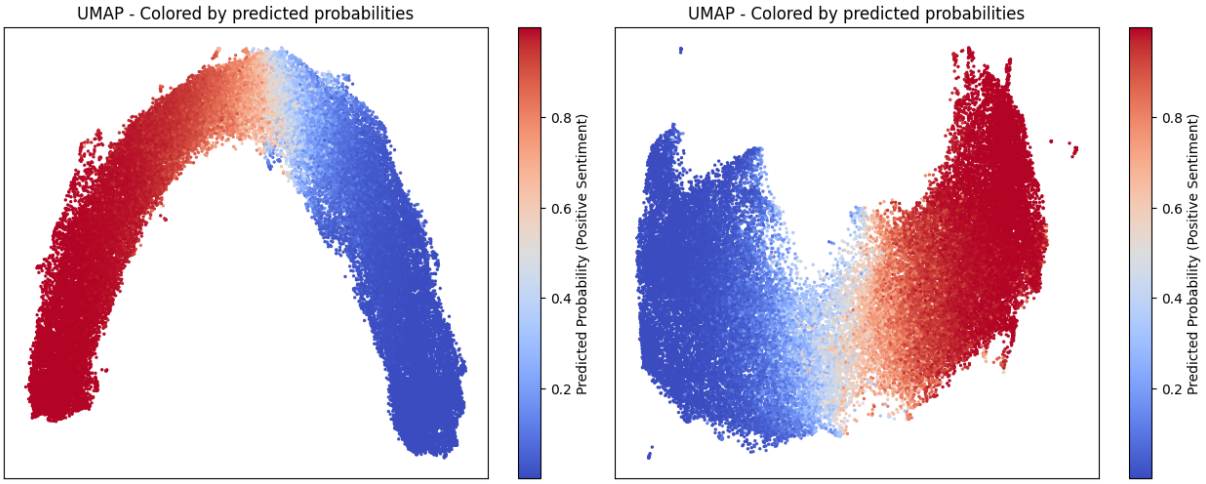
Figure 16: Distribution of probabilities

Upon studying the UMAP plots (Figures 17 and 18), the models' strength is further highlighted. Whether coloured by label or probability, two mostly distinct clusters are formed. This shows that the models can not only confidently differentiate between the two classes, but can do so with great confidence as well. However, when it comes to false predictions, the UMAPs show that the models are confidently incorrect in their predictions as well. Although each cluster contains outliers of the other group, the models mistakenly assign the same label to every apparent neighbour. Possible explanations, apart from poor finetuning, could include trouble in the differentiation of specific sentiments, such as sarcasm, or the aforementioned inherent ambiguity of language.



(a) BERT (b) DistilBERT

Figure 17: UMAP Coloured by Label



(a) BERT (b) DistilBERT

Figure 18: UMAP Coloured by Probability

The best hyperparameters were found to be the following:

	BERT	DistilBERT
Optimizer	AdamW	AdamW
Learning Rate	2.490311048764932e-05	4.2307241058454095e-05
Scheduler	cosine	cosine
Warmup Steps	475	475

After the last step, which was combining the training with the validation dataset and training the models again, the accuracy score reached on the testing dataset is

0.85650 by BERT and 0.84829 by DistilBERT. BERT achieved a higher score than DistilBERT, a result that was expected, as BERT has 44M more parameters and can therefore capture semantic nuances more effectively than DistilBERT.

As a final note, it would be ideal when finetuning the BERT model, that two Optuna studies be run, as was the case with DistilBERT. In this manner, all hyperparameters would be tuned, instead of simply using results found in the DistilBERT study. However, this was not achievable in a timely manner due to hardware limitations. Moreover, some experimentation with freezing layers should be included. This way, the pretrained model could be fully utilized, without being altered during the training process, and overfitting would be reduced, especially for the BERT model that has a large number parameters that might suffer during finetuning.

4.2. Comparison with the first and second projects

Comparatively to the first two projects, both the BERT and DistilBERT models perform better, while still having room for improvement, but required a more sophisticated approach in their optimization. This is to be expected, as transformers provide a plethora of options regarding their hyperparameters, as well as regularization and optimization methods. This wealth of options, however, introduces a need for more computational power. The Optuna optimization step suffered most from this; 30 trials were called for DistilBERT and only 5 for BERT, but they required 4 and 14 hours respectively to be completed when ran locally (on an RTX2060 GPU).

Overall, while their development is more time consuming and computationally heavy, it gives the opportunity to create a personalized model for the task at hand, that will correctly classify the data with great accuracy.

5. Bibliography

References

[1] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[2] Hugging Face Documentation. Learning Rate Schedules (PyTorch), 2024. Accessed: 2025-06-03.

[3] Hugging Face Documentation. Tokenizer, 2024. Accessed: 2025-06-03.

[4] Nefeli Gkouti, Prodromos Malakasiotis, Stavros Toumpis, and Ion Androutsopoulos. Should i try multiple optimizers when fine-tuning pre-trained transformers for nlp tasks? should i tune their hyperparameters?, 2024.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Aliyah Kurniasih and Lindung Parningotan Manik. On the role of text preprocessing in bert embedding-based dnns for classifying informal texts. *Neuron*, 1024(512):256, 2022.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[8] Chris McCormick. Smart Batching Tutorial - Speed Up BERT Training, 2020. Accessed: 2025-06-03.

[9] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[10] PyTorch Team. PyTorch, 2024. Accessed: 2025-05-04.