University of Athens
Department of Informatics and Telecommunications

Deep Learning for NLP

Student name: *Ilias Marras*
sdi: 1115201800111

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

Contents

1. Abstract

The task of this project is to develop a sentiment classifier by fine-tuning the pre-trained models Bert and distilBert. We will use the english language Twitter dataset. To tackle it, we analyze the data, perform preprocessing and then train and fine-tune our models to achieve the best performance.

2. Data processing and analysis

2.1. Pre-processing

Data cleaning and pre-processing are important steps that significantly affect our model's performance.
Since DistilBERT is a smaller version of Bert and mimics its behavior, applying the same preprocessing steps that work for Bert to DistilBert helps ensure optimal performance.
Based on my initial research and experimental results, Bert is a robust model that benefits from minimal pre-processing. Since Bert is trained on raw text and can handle elements like emojis, punctuation, and stop words, I applied fewer pre-processing steps compared to our previous projects. Additionally, techniques such as stemming or lemmatization led to a decrease in our models' performance, so I chose not to use them.
Specifically, the methods I used:

- remove URLs & html
- replace mentions with the token "@user"
- remove extra whitespace

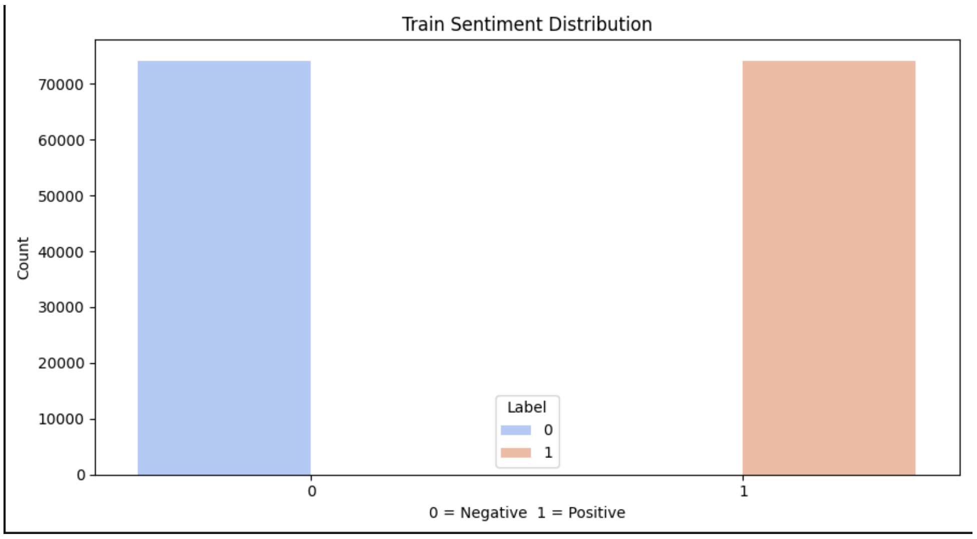
2.2. Analysis

For the analysis, I followed the same approach as in our previous projects, since the dataset remains the same. We try to get some initial info on the type of data that we have. To achieve that, we use plots and Wordclouds. Specifically:

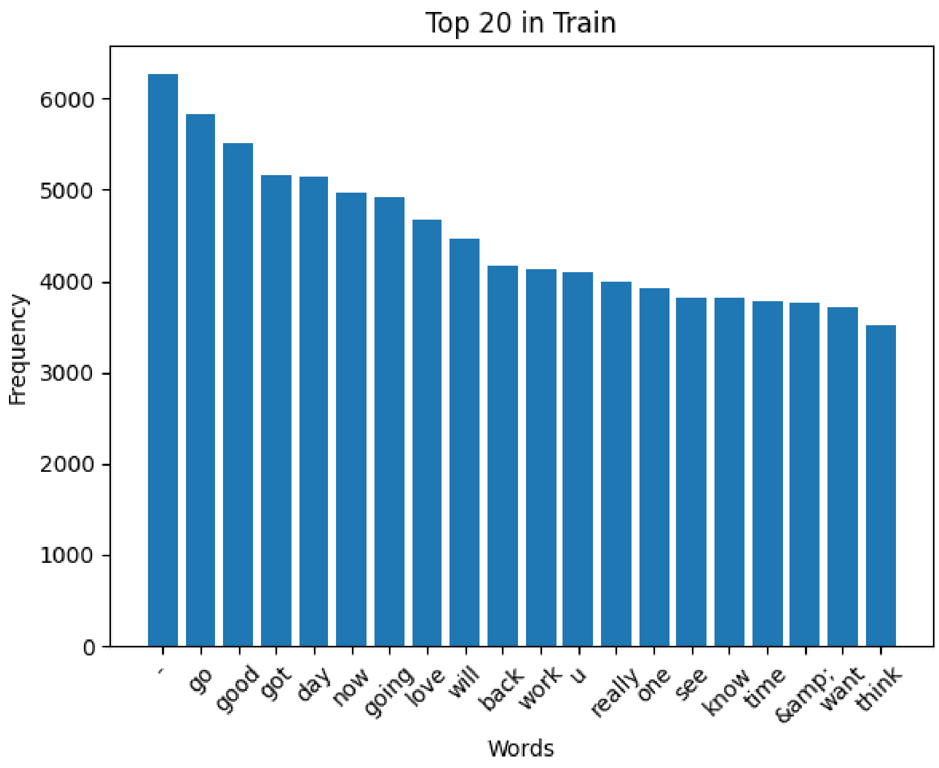
- plot for Sentiment Distribution
- 20 most used words
- WordCloud with the most frequent words

Data analysis was done for all the 3 datasets. To keep the report concise, I present here the plots for the train dataset only. Plots for the rest datasets can be found on the Bert ipynb notebook.

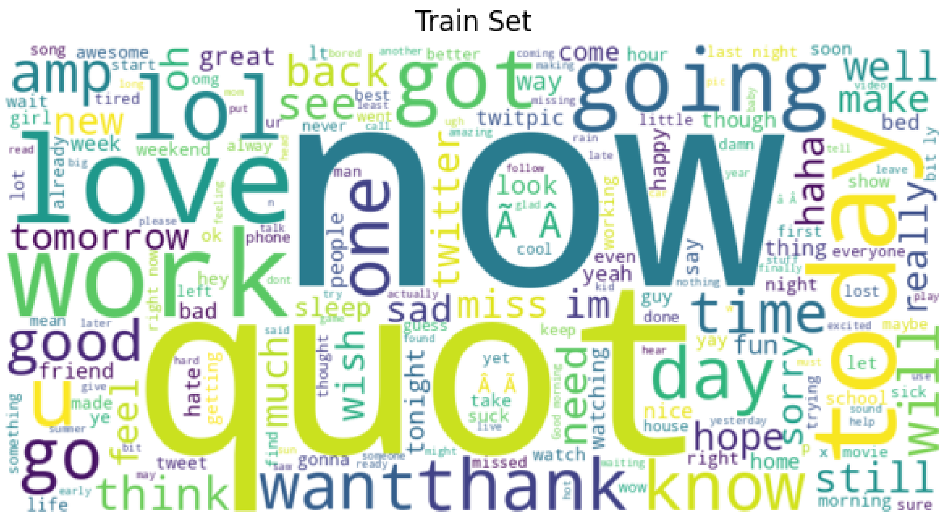
2.2.1. Sentiment Distribution.



2.2.2. Most Frequent Words.



2.2.3. Wordcloud.



2.3. Data partitioning for train, test and validation

For this project, data were already partitioned into training, validation, and test datasets. We used train data for training our model, validation data for validating and test data for testing our model's performance.

2.4. Vectorization

The same vectorization techniques were used for both Bert and distilBert models.
To convert our data into numerical input, we used the **bert-base-uncased** and the **distilbert-base-uncased** tokenizers from Hugging Face.
Below are the tokenizer parameters that I used for my final models:

- adding special tokens (e.g. [CLS] at the beginning)
- truncating sequences longer than max length (128 for us)
- padding to max length for shorter sequences

Then, the output of the tokenizer(input ids, attention mask) is fed into our Bert and distilBert models.

3. Algorithms and Experiments

3.1. Experiments

Initially, I adopted a brute-force approach to test how our models perform on our data. For the vectorization, I used the default tokenizer parameters - no padding, no truncation, no max sequence length. Then, trained the models for up to 4 epochs with a learning rate of 2e-5, a batch size of 64 and without applying weight decay.
The above provided a descent starting point in terms of performance, but led to significant overfitting, so the first thing that I did was to reduce the epochs to 2. As Bert is a strong and robust model, trained on raw data, I also tried to minimize the pre-processing steps and keep it as minimal as possible by only replacing mentions, removing URLS/HTMLS and extra whitespace. As like the previous projects, changing the pre-proccessing had a strong impact on our models' performance, since it led to an increase of about 0.01% in models' accuracy.
Then, I continued to experiment with the vectorization and the hyperparameters tuning. For vectorization, I added special tokens, truncation and padding, as mentioned in section 2.4. These adjustments improved the input quality and resulted in a modest accuracy improvement of approximately (0.005%)
Finally, I experimented with different hyperparameters, including learning rate, epsilon, weight decay, ADAM optimizer and also added linear schedule warm up instead of a steady one. For detailed results and configurations, please refer to Tables 1 and 2.

Notes: Since the two models are architecturally similar, I followed the same experimental procedure for both. Techniques that resulted in the best results for BERT were also applied to DistilBERT and vice versa.

Trial	Train Acc	Score (Val acc)
1	0.9521	0.8406
2	0.9034	0.8430
3	0.8877	0.8480
4	0.8793	0.8502
5	0.8713	0.8530

Table 1: Trials-Bert

Trial	Train Acc	Score (Val acc)
1	0.8890	0.8390
2	0.8768	0.8406
3	0.8790	0.8485

Table 2: Trials-DistilBert

Trial	Learning Rate	Epochs	Weight Decay	Batch Size	Optimizer
1	1e-4	4	-	128	AdamW
2	3e-5	3	-	64	Adam
3	2e-4	2	0.02	32	Adam
4	2e-5	2	0.01	128	AdamW
5	2e-5	2	0.01	128	AdamW

Table 3: Trial Parameters – BERT

Trial	Learning Rate	Epochs	Weight Decay	Batch Size	Optimizer
1	2e-5	4	-	64	Adam
2	3e-5	2	0.01	128	Adam
3	5e-5	2	0.01	128	AdamW

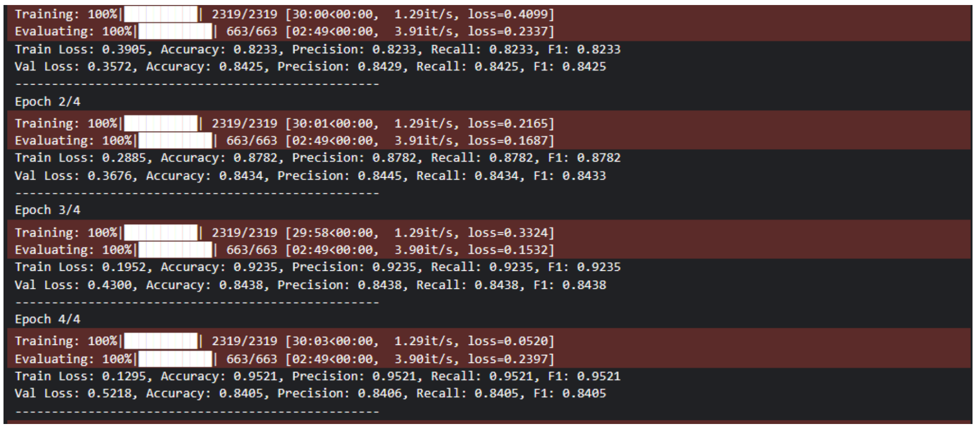
Table 4: Trial Parameters – DistilBERT

3.1.1. Table of trials.

Note: I conducted multiple trials, focusing on the most impactful ones. The tables above summarize key experiments done after completing the vectorization and pre-processing stages.

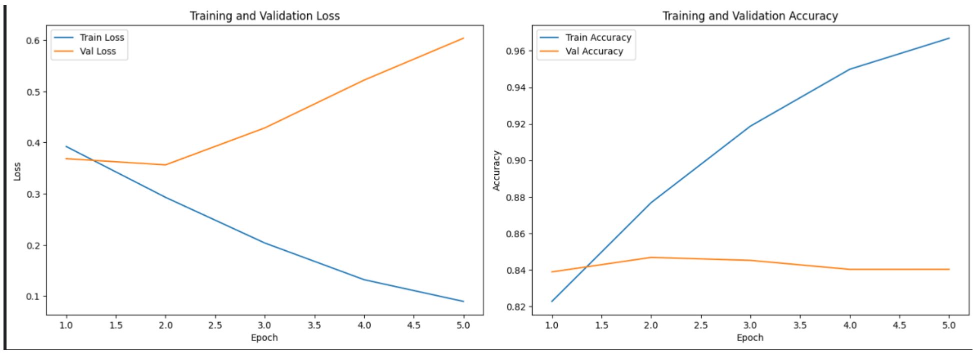
3.2. Hyper-parameter tuning

For the Hyper-parameter tuning I conducted multiple experiments to conclude to the best hyperparameters for my models, as mentioned above. I noticed that what worked well for the Bert model worked as well for the DistilBert, which makes sense due to their similar architecture. My experiments in detail are shown in previous sections.
From my initial experiments, I noticed a significant amount of overfitting for both of my models, especially Bert. As shown in the screenshot below, after 2 epochs we notice overfitting for our Bert model, since train accuracy inreases while validation accuracy drops.
Model: Bert | Epochs: 4 | Learning rate: 1e-4



```
Training: 100%|██████████| 2319/2319 [30:00<00:00,  1.29it/s, loss=0.4099]
Evaluating: 100%|██████████| 663/663 [02:49<00:00,  3.91it/s, loss=0.2337]
Train Loss: 0.3905, Accuracy: 0.8233, Precision: 0.8233, Recall: 0.8233, F1: 0.8233
Val Loss: 0.3572, Accuracy: 0.8425, Precision: 0.8429, Recall: 0.8425, F1: 0.8425
--------------------------------------------------
Epoch 2/4
Training: 100%|██████████| 2319/2319 [30:01<00:00,  1.29it/s, loss=0.2165]
Evaluating: 100%|██████████| 663/663 [02:49<00:00,  3.91it/s, loss=0.1687]
Train Loss: 0.2885, Accuracy: 0.8782, Precision: 0.8782, Recall: 0.8782, F1: 0.8782
Val Loss: 0.3676, Accuracy: 0.8434, Precision: 0.8445, Recall: 0.8434, F1: 0.8433
--------------------------------------------------
Epoch 3/4
Training: 100%|██████████| 2319/2319 [29:58<00:00,  1.29it/s, loss=0.3324]
Evaluating: 100%|██████████| 663/663 [02:49<00:00,  3.90it/s, loss=0.1532]
Train Loss: 0.1952, Accuracy: 0.9235, Precision: 0.9235, Recall: 0.9235, F1: 0.9235
Val Loss: 0.4300, Accuracy: 0.8438, Precision: 0.8438, Recall: 0.8438, F1: 0.8438
--------------------------------------------------
Epoch 4/4
Training: 100%|██████████| 2319/2319 [30:03<00:00,  1.29it/s, loss=0.0520]
Evaluating: 100%|██████████| 663/663 [02:49<00:00,  3.90it/s, loss=0.2397]
Train Loss: 0.1295, Accuracy: 0.9521, Precision: 0.9521, Recall: 0.9521, F1: 0.9521
Val Loss: 0.5218, Accuracy: 0.8405, Precision: 0.8406, Recall: 0.8405, F1: 0.8405
--------------------------------------------------
```

We can also notice that on the learning curves for our distilBert model:
Model: DistilBert | Epochs: 5 | Learning rate: 1e-4



To reduce overfitting, we choose less aggressive learning rate values for our Bert model and reduce the number of epochs to 2.
For our DistilBert model, we reduce the number of epochs to 2 while maintaining an aggresive learning rate, since that combination results to the best performance.
Below are the optimal hyperparameters identified for our models:

3.2.1. Best Hyper-parameters: Bert.

- Batch size: 128
- Optimizer: AdamW
- Learning rate: 2e-5
- Eps: 1e-8
- Weight decay: 0.01
- Epochs: 2
- Scheduler: linear_schedule_with_warmup

3.2.2. Best Hyper-parameters: DistilBert.

- Batch size: 128
- Optimizer: AdamW
- Learning rate: 5e-5
- Eps: 1e-8
- Weight decay: 0.01
- Epochs: 2
- Scheduler: linear_schedule_with_warmup

3.3. Optimization techniques

For optimization techniques, the ones that helped me boost my models' performance was finding the optimal learning rate, adding weight decay, a scheduler with linear warm up and finally changing the pre-processing steps by keeping only the most important ones.
Since Bert training requires a big amount of time (approximately 30 minutes for each epoch) I was not able to perform an Optuna Optimization for the best Hyper-parameters like my previous projects.

3.4. Evaluation

To evaluate our models, we will use the below scores:

- Accuracy: the proportion of all classifications that were correct
- Recall: the proportion of all actual positives that were classified as positives
- Precision: the proportion of all the model's positive classifications that are actually positive
- F1-score: combines precision and recall, balancing them

Metric	**Value**
Accuracy	0.8530
Precision	0.8530
Recall	0.8530
F1 Score	0.8530

Table 5: BERT performance

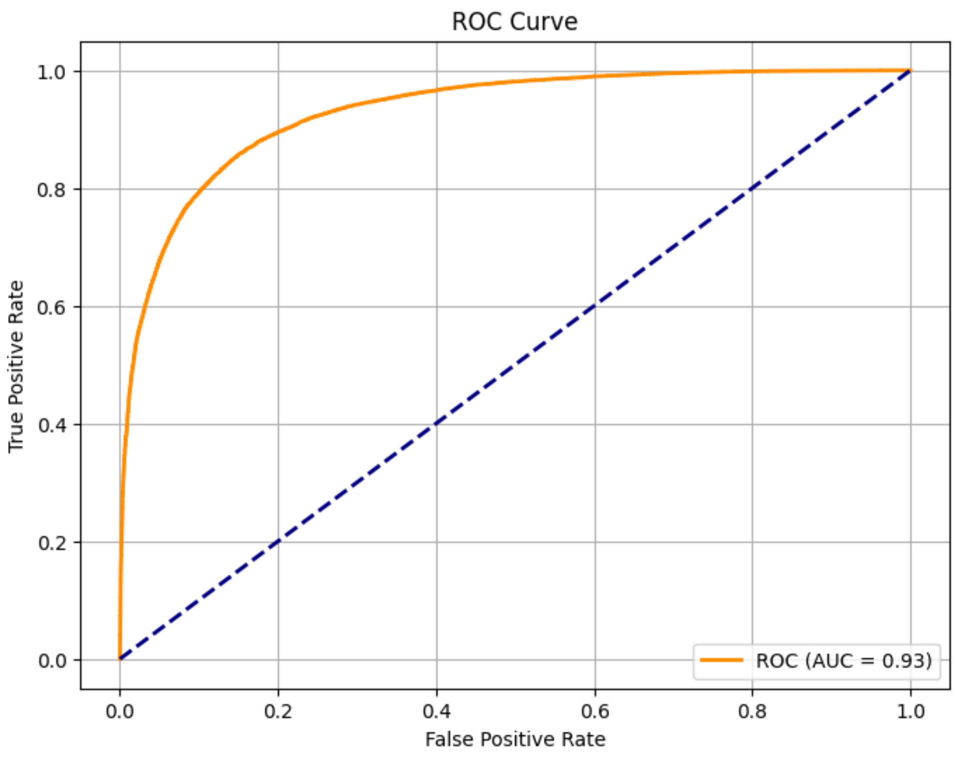
Metric	**Value**
Accuracy	0.8485
Precision	0.8486
Recall	0.8485
F1 Score	0.8485

Table 6: DistilBERT performance

As shown in the previous tables, the BERT model slightly outperformed the DistilBERT model across all evaluation metrics. This suggests that while both models performed well, BERT is stronger in classification accuracy and consistency, indicating its stronger architecture.

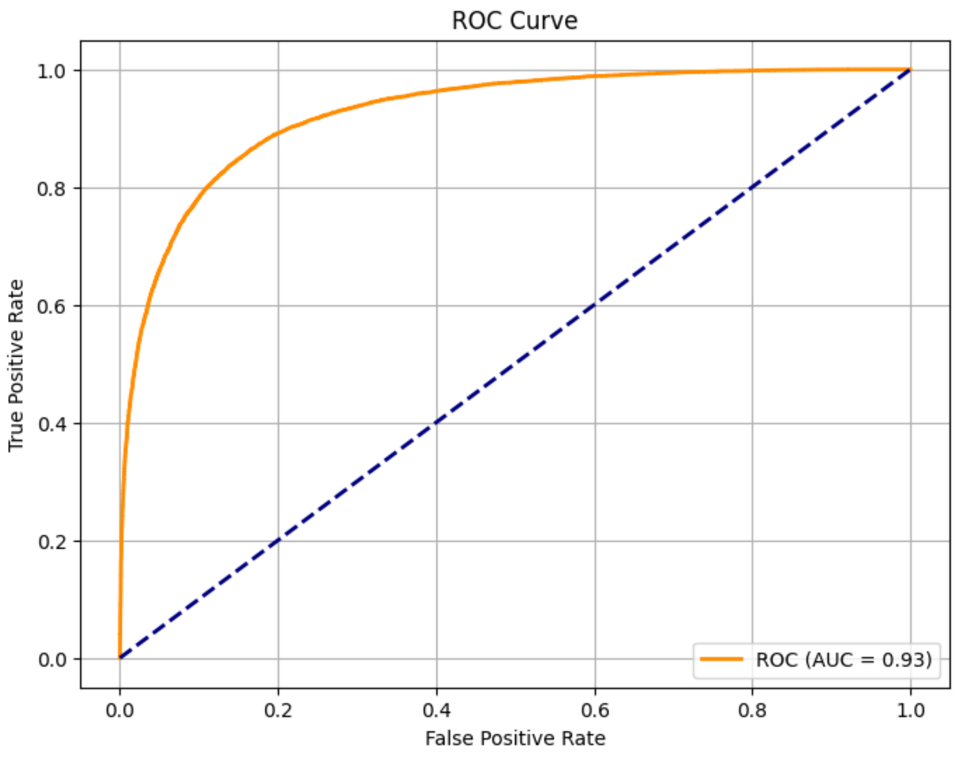
3.4.1. ROC curve.

BERT



A ROC score of 0.93 indicates that our BERT is highly effective at binary classification and distinguishing between positive and negative sentiment.

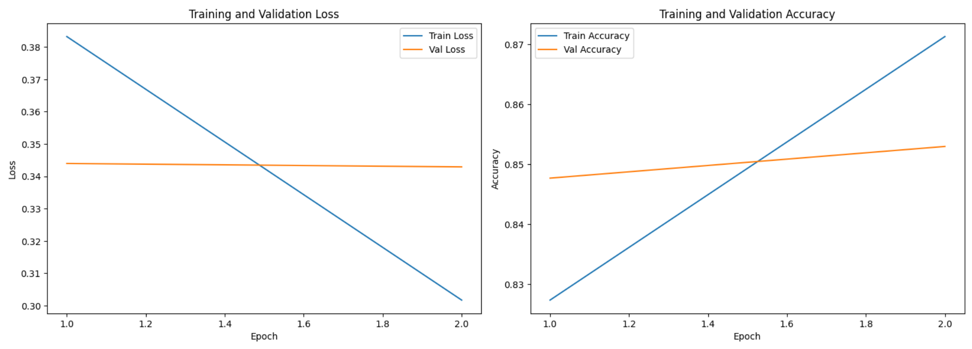
DistilBERT



The same goes for our DistilBERT model, since it achieves the same score.

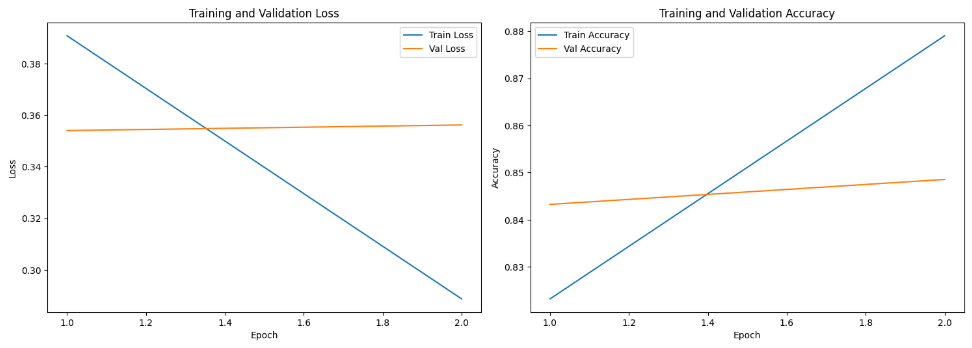
3.4.2. Learning Curve.

BERT



For our BERT model, we observe no signs of overfitting/underfitting during the first 2 epochs. The training loss decreases in every epoch, while validation loss has a slight downward trend. Validation accuracy keeps improving, indicating that the model generalizes well.
Trying to increase the number of epochs leads to signs of overfittingas mentioned in Section 3.2

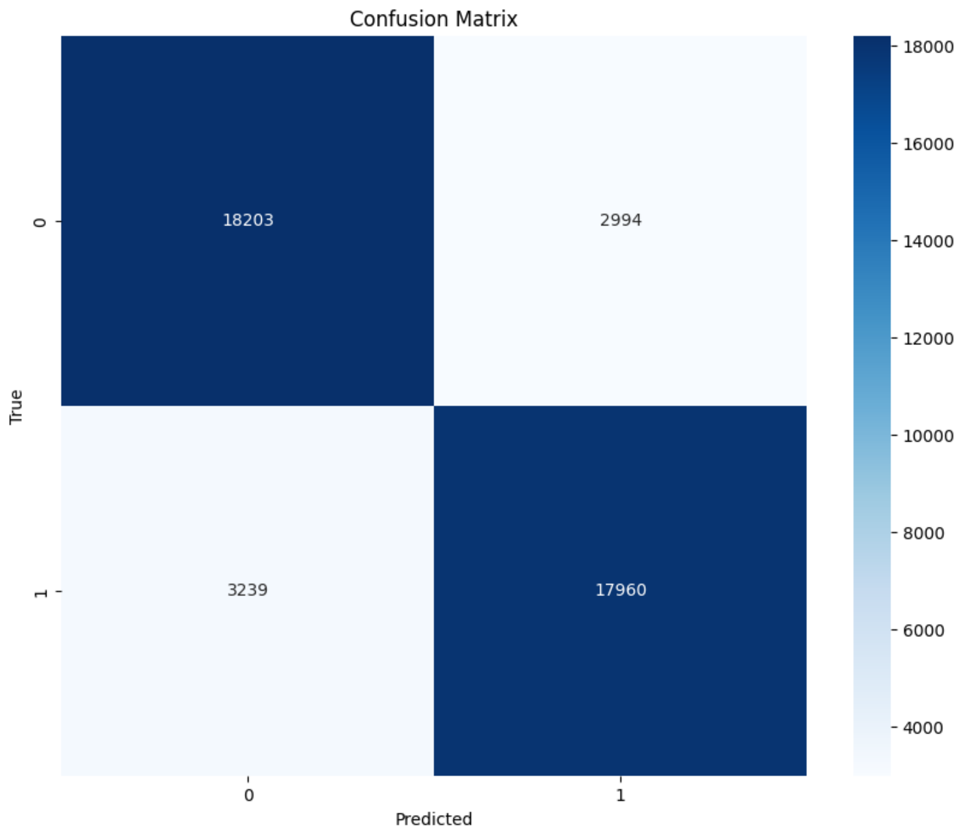
DistilBERT



The same goes for our DistilBERT model, since they produce a very similar Learning Curve.

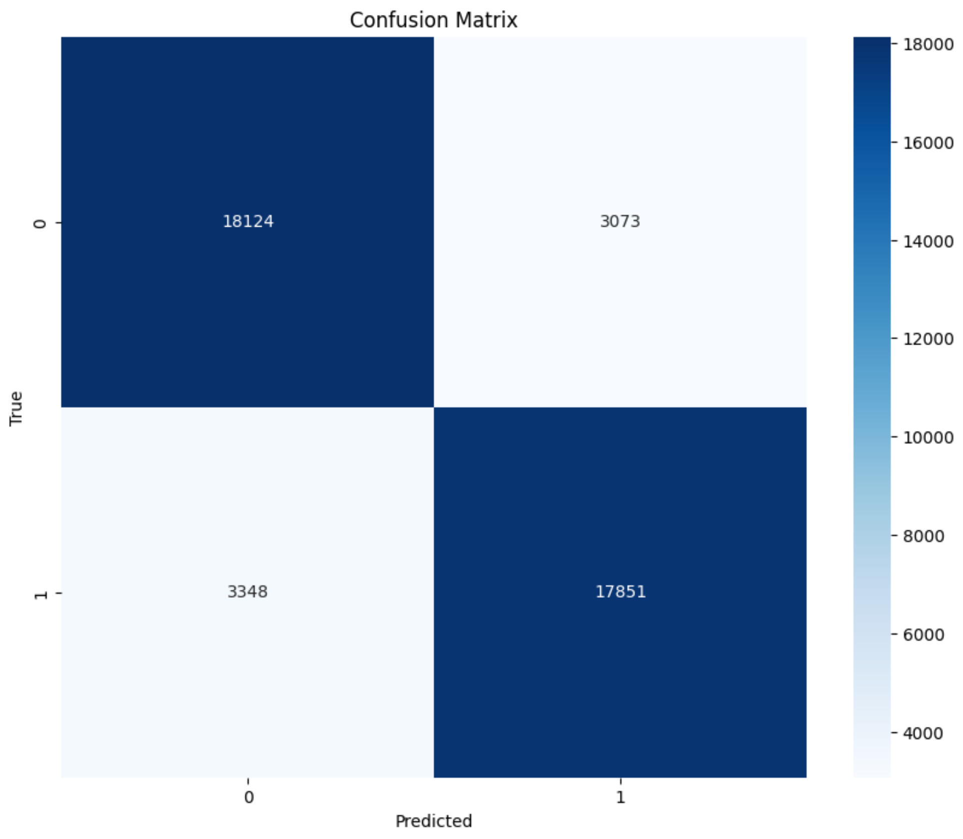
3.4.3. Confusion matrix.

BERT



The confusion Matrix for our BERT model reflect a strong and balanced performance, with relatively low false positive and false negative rates (2999, 3239) while maintaining good true negative and true positive values (18203, 17960).

DistilBERT



As for the DistilBERT model, while it still maintains a relatively strong performance between true positives/negatives and false classifications, it is slightly outperformed by the BERT model accros both classes.

4. Results and Overall Analysis

4.1. Results Analysis

My results so far suggest that our models demonstrate strong performance and generalize well, effectively distinguishing between positive and negative sentiment. As expected, BERT model outpeforms DistilBERT across all evaluation metrics (F1-score: 0.8530 vs. 0.8485), reflecting its stronger architecture.
An experiment that may benefit our models' performance is Optuna Optimization for finding the best Hyper-parameters. Since our Kaggle hours per week are limited, I prefered tweaking the models by experimenting with different values manually, but we could be able to achieve a little stronger performance. Finally, as shown in the Learning Curves, both DistilBERT and BERT are robust models that learn quickly, indicating that there is no need for additional training epochs.

4.1.1. Best trial.

BERT

- **Best Hyperparameters**: see Section 3.2.1
- **Kaggle Submission Score**: 0.85343
- **Accuracy**: 0.8530
- **Precision**: 0.8530
- **Recall**: 0.8530
- **F1-Score**: 0.8530

DistilBERT

- **Best Hyperparameters**: see Section 3.2.2
- **Kaggle Submission Score**: 0.84857
- **Accuracy**: 0.8485
- **Precision**: 0.8486
- **Recall**: 0.8485
- **F1-Score**: 0.8485

4.2. Comparison with the first project

Metrics (BERT vs LogisticRegression):

- Accuracy: 0.8530 VS 0.8009
- Precision: 0.8530 VS 0.80
- Recall: 0.8530 VS 0.80
- F1: 0.8530 VS 0.80

Metrics (DistilBERT vs LogisticRegression)):

- Accuracy: 0.8485 VS 0.8009
- Precision: 0.8486 VS 0.80
- Recall: 0.8485 VS 0.80
- F1: 0.8485 VS 0.80

Both DistilBERT and BERT outpeform the Logistic Regression model we fine-tuned in the first project. This is expected, since the Logistic Regression model has a significnatly simpler architecture, with much faster training times.

4.3. Comparison with the second project

Metrics (BERT vs DNN):

- Accuracy: 0.8530 VS 0.7884
- Precision: 0.8530 VS 0.7908
- Recall: 0.8530 VS 0.7842
- F1: 0.8530 VS 0.7875

Metrics (DistilBERT vs DNN):

- Accuracy: 0.8485 VS 0.7884
- Precision: 0.8486 VS 0.7908
- Recall: 0.8485 VS 0.7842
- F1: 0.8485 VS 0.7875

Similar to the previous comparison, both BERT models outpeform the DNN model of the previous project. This was expected since our previous DNN model had a <0.80 value across all metrics, while pre-trained BERT models has a strong capability in sentiment classification.
As of now, BERT seems to have the best performance across all different models we fine-tuned during the previous projects.

4.4. Kaggle Notebooks

- BERT
- DistilBERT

5. Bibliography

References

- BERT Guide
- Fine-Tuning BERT
- Padding and Truncation
- Transformers in Machine Learning
- Machine Translation
- Transformers
- BERT