UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Ergina Dimitraina*
sdi: <sdi2100031>

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.>
In this task, we were asked to implement a sentiment classifier using two pretrained models: BERT and DistilBERT. We began by exploring and cleaning the dataset, followed by splitting it into training, validation, and test sets, this process that was the same for both models. Then, we integrated the two models and conducted a series of experiments to optimize accuracy for each one. Finally, we presented the results using diagrams for better visualization.

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.>
In this step, the raw text data was cleaned and pre-processed to improve the quality and consistency of the input for the machine learning model. Proper data preprocessing is essential in any NLP task, as it directly impacts model performance and training efficiency. The following methods were applied:

HTML and Special Character Removal: Each text sample was first passed through the BeautifulSoup parser to remove any embedded HTML tags. Then, regular expressions were used to remove unwanted characters such as special symbols, brackets, and other non-alphanumeric content. This helped reduce noise in the dataset.

Lowercasing and Simplification: Although not explicitly shown, the regular expressions allowed alphanumeric characters and a limited set of punctuation, implicitly helping normalize the data. Text was kept in a relatively clean and uniform format to reduce vocabulary size.

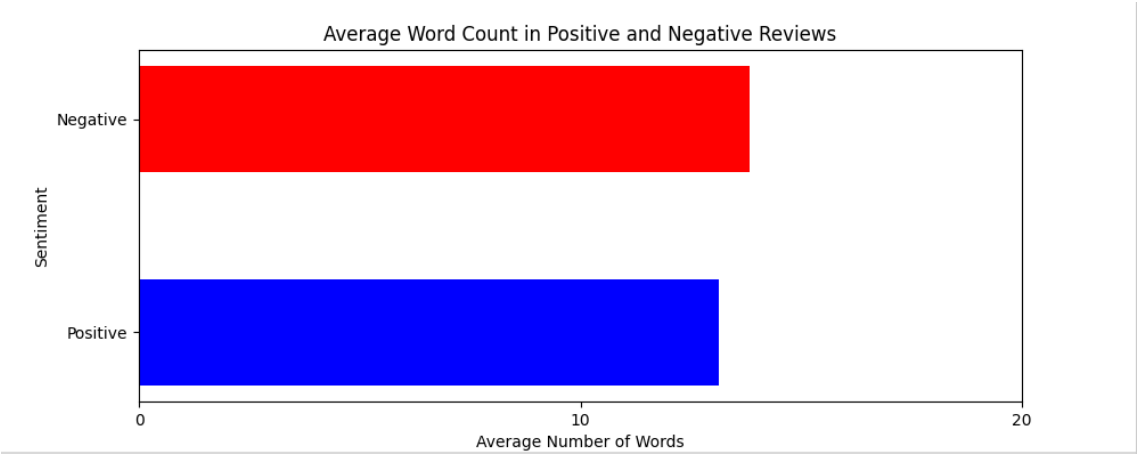
Handling Missing Values and Labels: Rows with missing labels were removed to ensure training consistency. The labels were also explicitly cast to integers to match the expected format for classification tasks.

2.2. Analysis

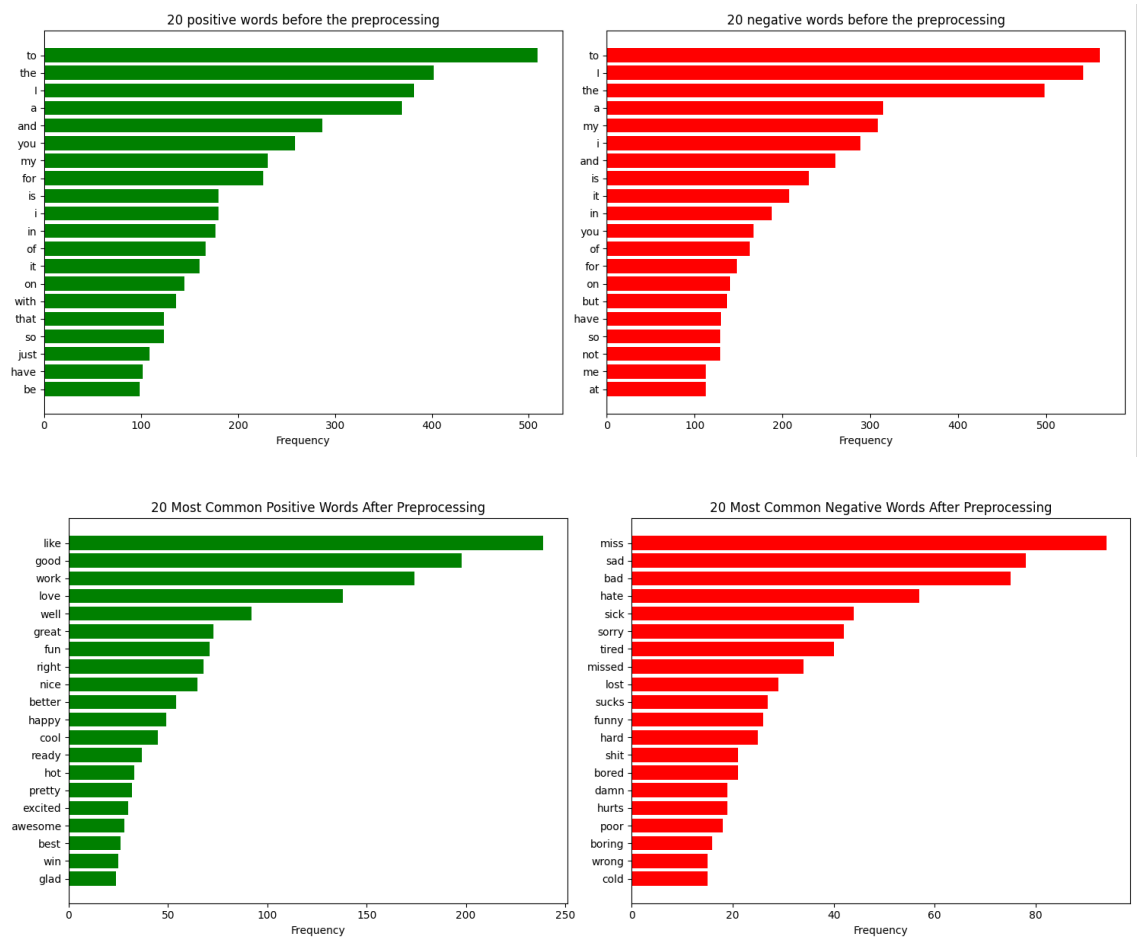
<In this step, you should also try to visualize the data and their statistics (e.g., word clouds, tokens frequency, etc). So the processing should be done in parallel with an analysis. >

In this section, the diagrams are used to visualize various statistics of the data. The same preprocessing techniques were applied to both models (BERT and DistilBERT).

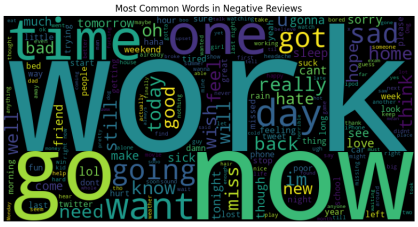
The first diagramm presents the average number of words in each sentiment.



The second set of diagrams presents the 20 most common positive and 20 most common negative words, both before and after preprocessing. These four diagrams help us observe which words tend to appear more frequently in the dataset prior to preprocessing, potentially leading to less accurate results. After preprocessing, we ensure that the majority of the dataset is 'clean' and contains words that contribute meaningfully to the sentiment classification process.



Word clouds provide a general overview of the words that most significantly influence our model. They serve as an effective visualization tool, helping us identify key terms that impact sentiment classification.



2.3. Data partitioning for train, test and validation

<Describe how you partitioned the dataset and why you selected these ratios>
In this project, we used sampling on the training data. Due to the large size of the dataset, training the model on the entire data would take more than 24 hours. Therefore, I chose to randomly sample the training data each time, which made it easier to observe the results and update the model efficiently.

The original dataset was divided into three subsets: training, validation, and test sets. This partitioning was done to effectively train the model, fine-tune hyperparameters, and evaluate generalization performance. The process was as follows: After sampling 2% of the original dataset for faster experimentation, 80% of the data was allocated to the training set and 20% to the test set. This ensured that the model could learn from the majority of the data while still being evaluated on unseen examples. From the training set, 10% was further split off to form a validation set.
This resulted in a final split of:
72% Training
8% Validation
20% Test
The validation set was used during training to monitor model performance and prevent overfitting by guiding decisions such as the number of epochs or early stopping. These ratios strike a balance between having enough data for learning while preserving separate datasets for model tuning and unbiased evaluation.

2.4. Vectorization

<Explain the technique used for vectorization>
In this project we have 2 vectorization techiniques BERT and DistilBERT
BERT:
DistilBERT: we used the DistilBERT tokenizer from the Hugging Face Transformers library, which create sparse and shallow representations, provides contextualized embeddings that capture deeper semantic relationships between words. The text is broken down into subword units using WordPiece tokenization, which helps handle out-of-vocabulary words more effectively. Each sequence is padded or truncated to a fixed maximum length (128 tokens in this case), ensuring uniform input size across all batches. The tokenizer converts each token into a corresponding numerical ID based

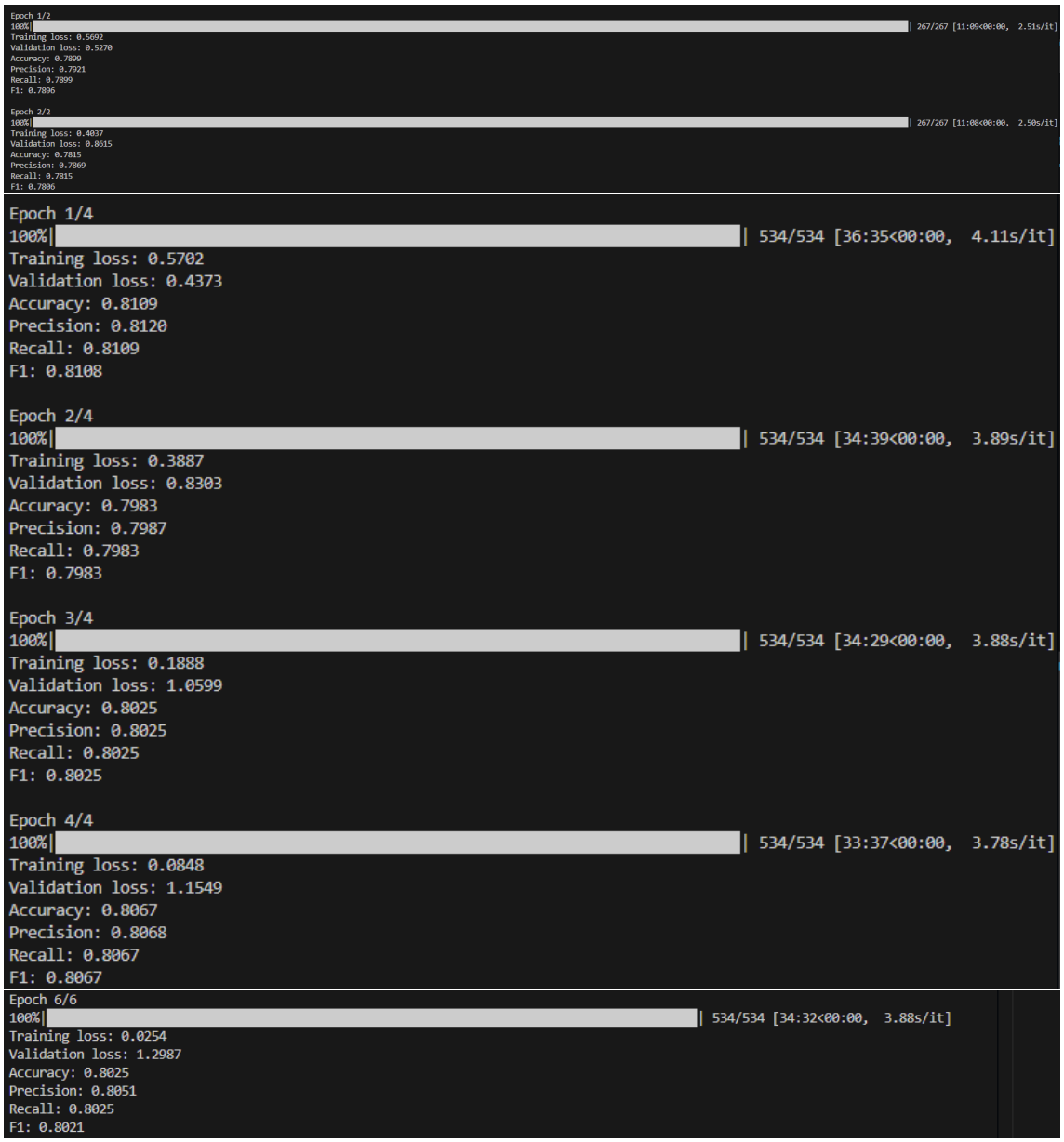
on the DistilBERT vocabulary. It also generates attention masks to differentiate between actual tokens and padding. The result of this process is a pair of tensors: one for input IDs and one for attention masks. These tensors are fed directly into the DistilBERT model for training and inference. This vectorization approach is powerful because it produces dense, context-aware embeddings that significantly improve model performance on NLP tasks.

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>

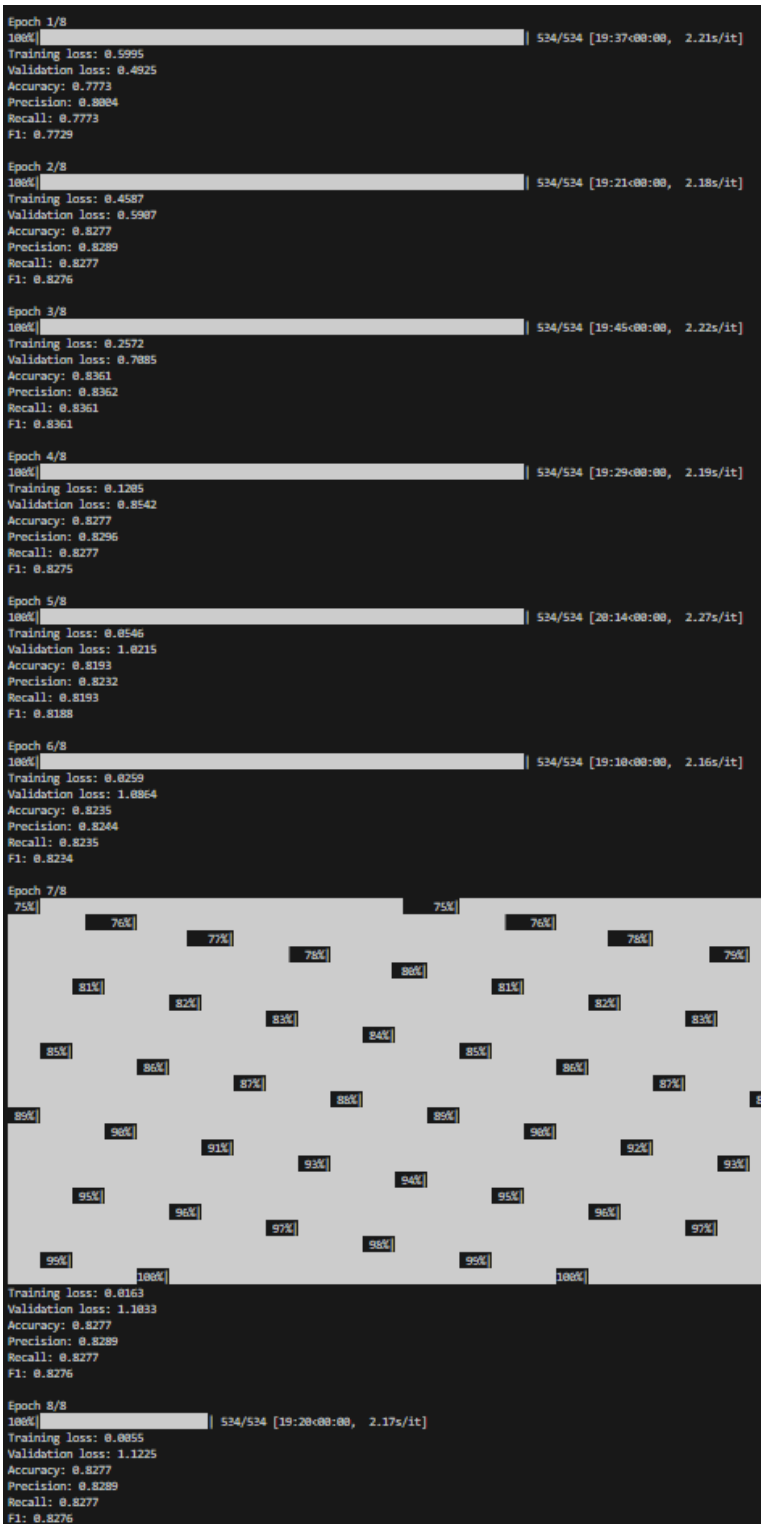
To tackle the sentiment classification problem, we followed a systematic experimental approach, starting from a basic brute-force pipeline and gradually introducing improvements through various techniques in preprocessing, training configuration, and optimization. The first experimental run used the full dataset without subsampling, minimal preprocessing, and a basic fine-tuning of the models using default parameters. However, this led to an extremely long training time (estimated over 20 hours per epoch) and GPU memory exhaustion, making it infeasible for iterative development. After the results of the brute force approach we tried dataset subsampling. We reduced training data to 2% of the original dataset. The reason behind this was to speed up experimentation and allow quick iteration on model training and evaluation. The result was that the training time became manageable (several minutes per epoch), and we were allowed to make corections while the accuracy is still high (80-85 percent). Then to improve the accuracy of the models more we made some preprocess improvments. We cleaned the raw text data using BeautifulSoup to strip HTML, removed special characters, and converted all text to a normalized format, this helped remove noise in the dataset, improving learning stability. Particularly improved precision in both positive and negative classes. After finishing with the preprocess we tried training configuration and epoch tuning. We tried several trainig epochs, as shown in the results below. The result was that the more data we have the more epochs we want to achieve good accuracy. Then we tried to reduce the batch sizes to reduce run time, this enabled training, but with slightly noisier gradient updates. Training still remained stable due to the learning rate scheduler. Finally we tried several maximun sequence lenghts for the tokenizers, but we ended up for max length 128 tokens, which enables faster and more effecttive training. Increasing the length led to out of memory errors.
BERT



DistilBERT

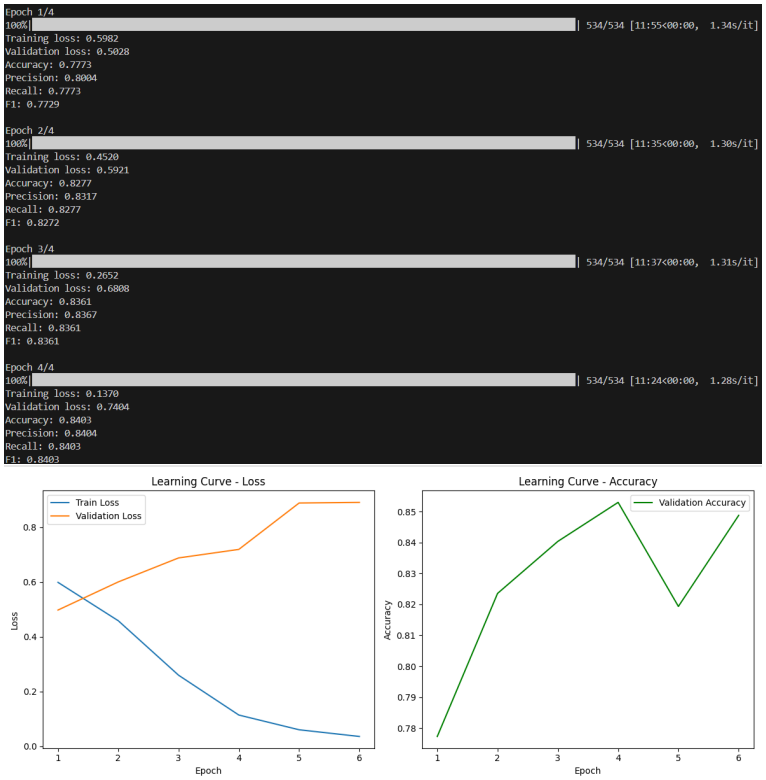
sample size 0.05

sample size 0.02



In the case below we observe that although the best accuracy is achived with 6 epochs, in the learning curve we observe overfitting.Train Loss (blue line) consistently decreases across all epochs, which indicates that the model is effectively learning patterns from the training data. Validation Loss (orange line) increases gradually and then plateaus between epochs 5 and 6. This is a clear indication of overfitting: the model continues to improve on the training set but performs worse on unseen validation data.Validation Accuracy (green line) shows initial improvement, peaking at epoch 4, followed by a noticeable drop at epoch 5, and then a slight recovery at epoch

6. This fluctuation confirms the pattern observed in the loss curves, the model begins to overfit after the 4th epoch. **Epoch 4 seems to be the optimal point where validation accuracy is highest and overfitting hasn't yet significantly impacted performance.**



3.2. Hyper-parameter tuning

<Describe the results and how you configured the model. What happens with under-over-fitting??>
Hyper-parameter tuning played a critical role in configuring the models to achieve balanced performance without overfitting or underfitting. While limited computational resources restricted an extensive grid search, several key hyper-parameters were manually selected and adjusted based on validation performance:

Configured Hyper-parameters:

Learning Rate: Set to 2e-5, which is commonly recommended for fine-tuning transformer models. Higher learning rates led to unstable training, while lower values slowed down convergence.

Batch Size: A small batch size of 4 was used during training due to memory limitations. Validation was done with a batch size of 8 to increase evaluation efficiency.

Epochs: The model was trained for 4 epochs, which was sufficient for convergence on the reduced dataset (2% of the original). More epochs led to signs of overfitting.

Max Sequence Length: Texts were truncated or padded to a maximum length of 128 tokens, balancing between context retention and computational efficiency.

Underfitting and Overfitting Observations: Underfitting: When using fewer epochs or too low a learning rate, the model failed to capture meaningful patterns from the data, resulting in both low training and validation performance.

Overfitting: After several epochs, training loss continued to decrease while validation performance plateaued or worsened, indicating overfitting. This was mitigated by limiting the number of epochs and using weight decay (AdamW optimizer) and learning rate scheduling.

The selected hyper-parameters resulted in a well-generalized model on the sampled dataset.

3.3. Optimization techniques

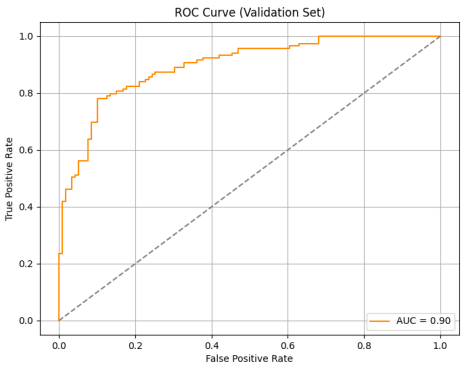
<Describe the optimization techniques you tried. Like optimization frameworks you used.>
To optimize the performance of the BERT and DistilBERT models during training, the following techniques and frameworks were used:

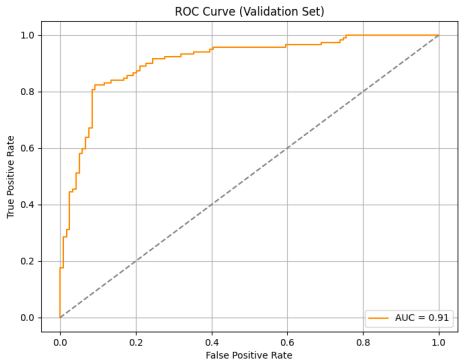
AdamW Optimizer: We used the AdamW optimizer, which is an improved version of the traditional Adam optimizer that decouples weight decay from the gradient update. This is particularly well-suited for transformer-based models like DistilBERT and helps prevent overfitting by applying regularization more effectively.
Learning rate: 2e-5
Epsilon: 1e-8 (for numerical stability)
Learning Rate Scheduler: A linear learning rate scheduler with warm-up was applied. This scheduler gradually increases the learning rate at the beginning (warm-up phase) and then decreases it linearly throughout training. This helps stabilize early training and improves convergence.
Gradient Clipping: To prevent exploding gradients, especially during backpropagation in deep models, we used gradient clipping with a maximum norm of 1.0. This ensures that gradients stay within a manageable range and training remains stable.
Batch Size and Epochs:
Training batch size: 4
Validation batch size: 8
Number of epochs: 4
These values were selected to balance training time and performance, especially given the hardware and time constraints. These optimization techniques were crucial in ensuring that the model converged efficiently while maintaining generalization performance on unseen data.

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>
<Provide and comment diagrams and curves>

3.4.1. ROC curve.

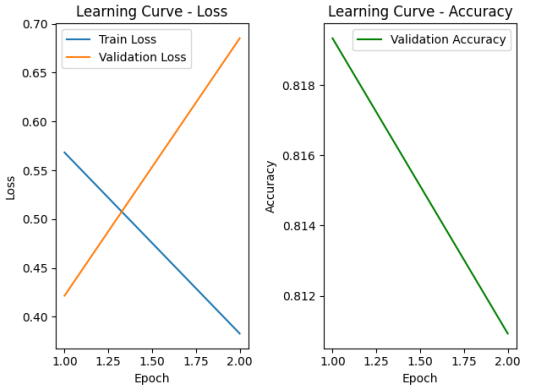


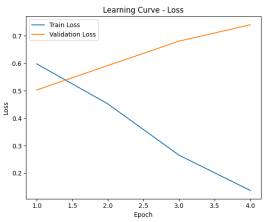


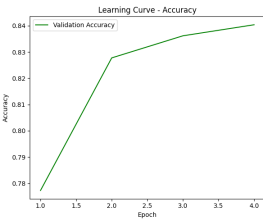
BERT DistilBERT

3.4.2. Learning Curve. **BERT**

On the learning curves(loss), we observe that the training loss decreases steadily across all epochs, indicating that the model is successfully learning from the training data. On the right plot (Accuracy), we can see that the validation accuracy improves consistently over all four epochs. This suggests that despite the rise in validation loss, the model's predictive performance is still increasing, which may be due to better confidence in correct predictions. However, the gap between training and validation losses is because the model is trained in small amount of data and if we continue to add epochs we may encounter overfitting so caution should be taken in continuing training beyond this point. Overall, the model shows strong learning behavior during the epochs.

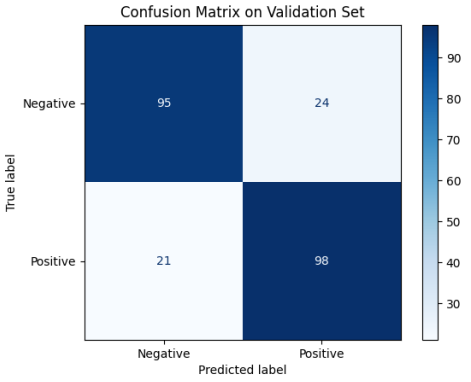




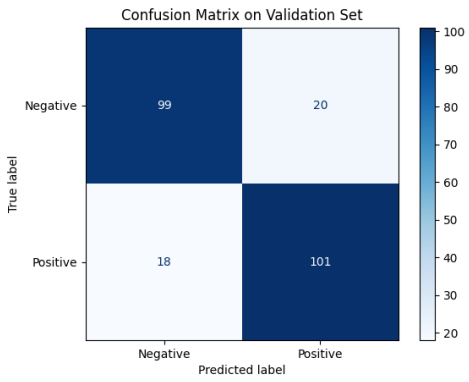


BERT DistilBERT

3.4.3. Confusion matrix.



BERT



DistilBERT

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?>

<Provide and comment diagrams and curves>

The results obtained so far demonstrate satisfactory performance from both models, with DistilBERT achieving slightly higher accuracy compared to BERT. This was somewhat expected due to its simpler architecture and faster training time, despite BERT being generally more powerful but also more computationally intensive.

It is important to note that all experiments and model tuning were conducted using only 2% of the training dataset. The hyperparameters were selected based on the best-performing trials on this small sample, under the assumption that similar trends would hold when scaling to the full dataset.

4.1.1. Best trial. <Showcase best trial>

BERT:

```
Epoch 1/2
100%| 534/534 [1:15:49<00:00,  8.52s/it]
Training loss: 0.5682
Validation loss: 0.4215
Accuracy: 0.8193
Precision: 0.8195
Recall: 0.8193
F1: 0.8193

Epoch 2/2
100%| 534/534 [37:47<00:00,  4.25s/it]
Training loss: 0.3827
Validation loss: 0.6852
Accuracy: 0.8109
Precision: 0.8111
Recall: 0.8109
F1: 0.8109
```

DistilBERT:

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?>

4.3. Comparison with the second project

<Use only for projects 3,4>
<Comment the results. Why the results are better/worse/the same?>

4.4. Comparison with the third project

<Use only for project 4>
<Comment the results. Why the results are better/worse/the same?>

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>