UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Maria-Malamati Papadopoulou*
sdi: sdi2200134

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

Contents

1. Abstract

The objective of this assignment is to develop a sentiment classification model using two advanced transformer-based architectures: **BERT** and **DistilBERT**. The models are fine-tuned on a Twitter dataset annotated with binary sentiment labels—positive (1) and negative (0). Given the noisy and informal nature of social media text, the approach begins with comprehensive preprocessing that includes normalization of slang, correction of common spelling errors, removal of mentions, hashtags, and URLs, and tokenization specifically tailored to tweets. Each preprocessed tweet is transformed into a fixed-length sequence of token IDs using the appropriate HuggingFace tokenizer (BERT or DistilBERT), ensuring compatibility with the pretrained language models. These inputs are then fed into a PyTorch-based classification pipeline built on top of the respective transformer, where the final hidden representations are used to predict sentiment classes. Model evaluation is carried out using a suite of metrics, accuracy, precision, recall, and F1-score, on both the validation and test sets, with training progress visualized through detailed learning and loss curves, ROC curves and confusion matrices. This pipeline ensures both performance and generalization, while maintaining reproducibility via fixed random seeds.

Links to my Kaggle notebooks

- BERT:
  https://www.kaggle.com/code/mairapapadopoulou/sdi2200134-3
- DistilBERT:
  https://www.kaggle.com/code/mairapapadopoulou/sdi2200134-3-distilbert

2. Data processing and analysis

2.1. Pre-processing

Before applying the BERT and DistilBERT Tokenizers and proceeding with model training, it was essential to perform careful pre-processing of the input text. Effective pre-processing improves the consistency and semantic clarity of the data, helping the transformer-based models to extract more meaningful patterns from noisy, informal content such as tweets. The following steps were implemented:

1. Lowercasing: Converting all text to lowercase ensures that words like "The" and "the" are treated the same, preventing unnecessary duplication in the vocabulary and promoting more consistent word embeddings.
2. Tweet-specific Tokenization: Tweets were tokenized using the TweetTokenizer from NLTK, which is specifically designed to handle social media text. This step correctly processes Twitter-specific features such as hashtags, mentions, emojis, and emoticons, preserving the linguistic structure while splitting the text into semantically meaningful units.
3. Contractions: Contractions, like "isn't" to "is not" were expanded to their full forms. This increases clarity in the data and ensures that pre-trained GloVe embeddings can correctly map words without missing representations for shortened forms.

4. Removing mentions, hashtags and links: Eliminating Twitter mentions (@user) for complete anonymization, hashtags and URLs to focus on the actual tweet content due to analysis' results.

5. Removing Special Characters and Numbers: Filtering out punctuation, non-ASCII characters, and numbers to reduce noise in the data.

6. Normalizing repeated characters and spaces: Twitter language often includes elongated words for emphasis like "plzzzz" instead of "please". I normalized these cases by reducing repeated characters to a single or standard form, for example, "plzzzz" to "plz". Similarly, redundant spaces were standardized to a single space to preserve the structural uniformity of the data.

7. Slang normalization: Common internet slang and abbreviations were mapped to their standard English equivalents. This improves the semantic interpretability of the tweets and ensures that the pre-trained embeddings can correctly process the normalized words.

Each of these pre-processing steps plays a critical role in minimizing noise, reducing vocabulary sparsity, and improving generalization. In combination, they contribute to constructing a high-quality input representation that enables the BERT-based models to focus on the underlying sentiment rather than artifacts of informal language or typographical variation.

2.2. Analysis

To gain a comprehensive understanding of the training dataset, I first computed the number of unique words across the training data, revealing approximately 234,492 distinct features (see Figure 1). This high vocabulary size highlights the linguistic diversity typical of user-generated content like tweets, which often include slang, typos, informal contractions, and creative spelling variations.
Next, I examined the class distribution within the training set. Out of a total of 148,388 tweets, the distribution was approximately even, with positive and negative tweets each comprising around 50% of the data (see Figure 2). As a result, the training dataset exhibits a balanced class distribution and this ensures that the model does not suffer from class imbalance, which could have led to biased predictions favoring the majority class.
Similarly, analysis of the validation set, consisting of 42,396 tweets, also indicated a balanced 50/50 split between positive and negative sentiments (see Figure 3). The balanced nature of both the training and validation datasets ensures that the model's performance metrics will be more reliable and not artificially inflated due to a dominant class.
These insights from the previous project helped refine the preprocessing steps and provided an initial understanding of how sentiment is expressed in the dataset. EDA, which was conducted in first project, exclusively on the training dataset to prevent any potential data leakage from the validation and test dataset, thereby ensuring the integrity of the evaluation process and maintaining unbiased performance assessment. Additionally, since the model is trained solely on the training dataset, focusing the analysis on this subset ensures that the extracted insights and preprocessing choices

are directly relevant to what the model will learn. This makes the EDA more representative and meaningful for guiding model development.

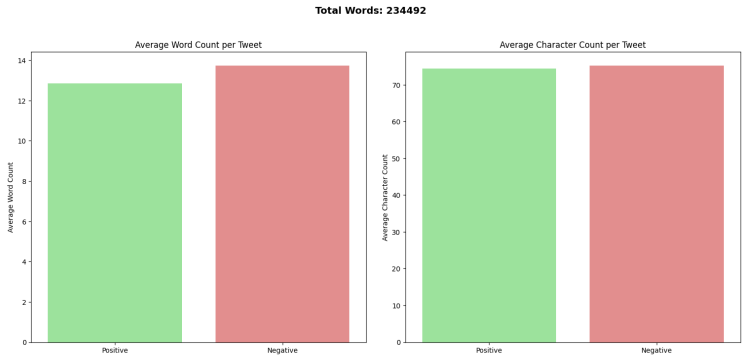

Figure 1: Average Word and Character Count per Tweet

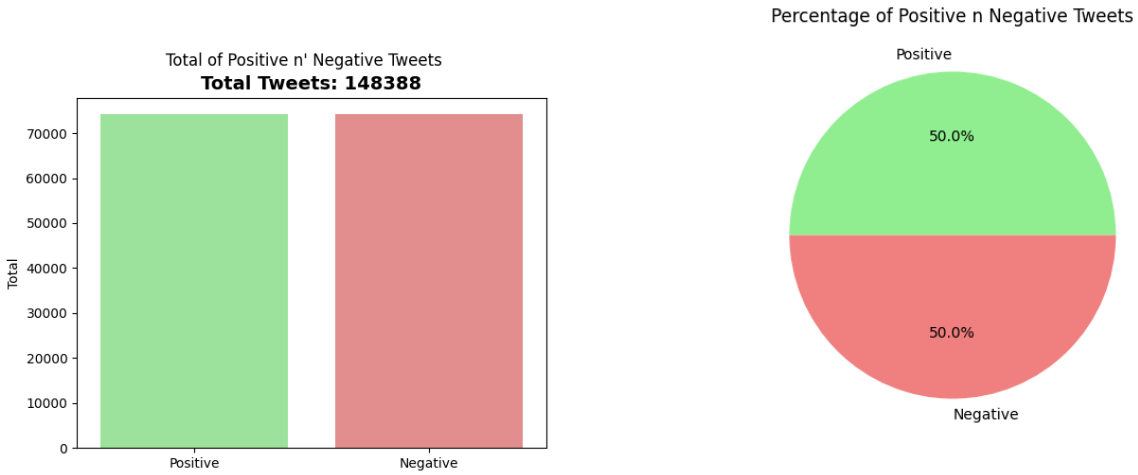

Figure 2: Comparison of Positive and Negative Tweets in Training Set

Lastly, to determine an appropriate max_length value for tokenization, I conducted an analysis of the token distribution across all tweets in the training dataset. As shown in Figure 4, the vast majority of tweets contain fewer than 60 tokens when processed using the pretrained tokenizer (BERT/DistilBERT). In particular, the distribution is heavily skewed toward shorter texts, with very few examples exceeding 60 tokens. Selecting an appropriate max_length is critical for both computational efficiency and model performance. Setting this value too low risks truncating meaningful content, potentially leading to underfitting. On the other hand, using an excessively large max_length increases memory usage, introduces unnecessary padding, and may promote overfitting by encouraging the model to focus on rare long sequences. Based on the histogram, which reveals that tweets longer than 60 tokens are extremely rare (fewer than 10 occurrences per token count beyond 60), I chose max_length = 60 as a balanced cutoff. This choice preserves the full content of the vast majority of tweets while minimizing padding and memory overhead.

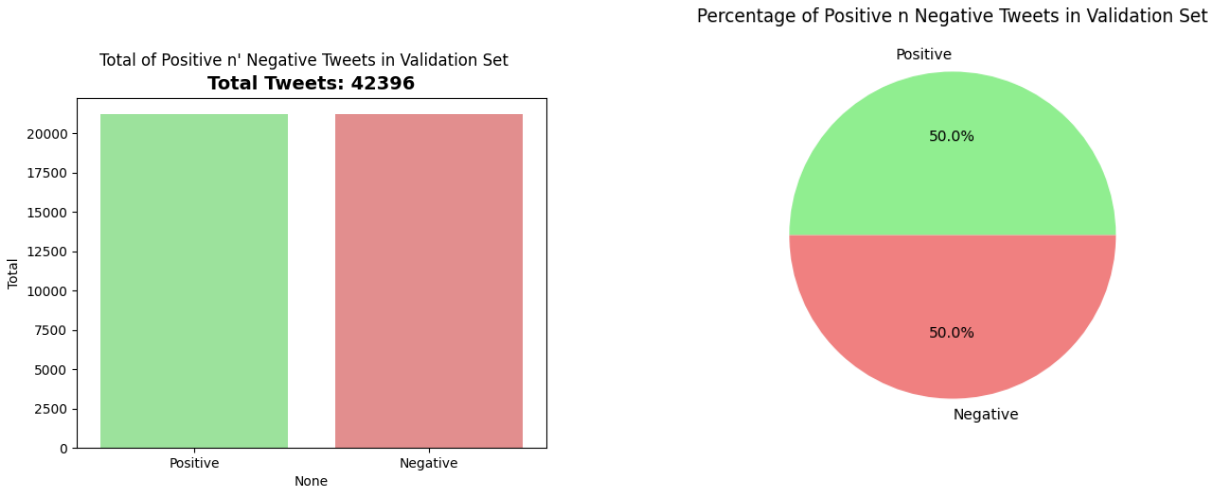

Figure 3: Comparison of Positive and Negative Tweets in Validation Set

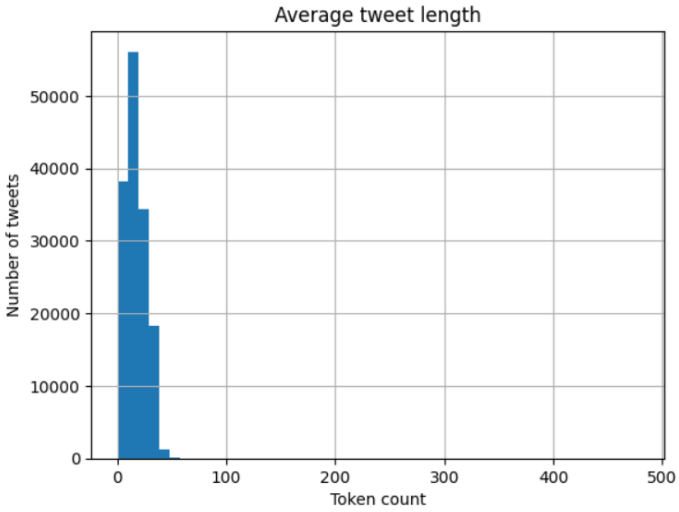

Figure 4: Average Tweet Length

2.3. Data partitioning for train, test and validation

To effectively train and evaluate the sentiment classification model, the dataset was partitioned into three subsets: training set, validation set, and test set, as it was shown in the lectures. The dataset was split as follows:

1. Training Set (X_train, y_train): This set contains 148,388 tweets, as identified during the Exploratory Data Analysis (EDA). It is used to fit the model, allowing it to learn patterns, word associations, and sentiment relationships from labeled examples. A sufficiently large training set ensures that the model can generalize well without underfitting.

2. Validation Set (X_val, y_val): This subset contains 42,396 tweets and is employed throughout training to evaluate the model's intermediate performance. It is used for hyperparameter tuning, monitoring overfitting, and guiding decisions such as learning rate adjustments, dropout settings, and early stopping. Since the validation set is not seen during training, it provides a realistic proxy for generalization.

3. Test Set (X_test): The test set consists of a separate, unseen portion of the data that was not utilized during model training or validation. It serves as the final benchmark to evaluate the model's real-world performance and assess its ability to generalize to completely new inputs.

This fixed partitioning setup ensures that each phase of the modeling process, training, tuning, and evaluation, is carried out independently and without data leakage, thereby maintaining the integrity of the performance metrics.

2.4. Vectorization

Vectorization refers to the process of converting raw textual input, tweets specifically, into numerical representations that can be processed by deep learning models. Given that I employed transformer-based architectures (BERT and DistilBERT), the vectorization process is handled via their respective pretrained tokenizers provided by the HuggingFace library. Before vectorization, each tweet was preprocessed (as described in Section 2.1) and then passed through the appropriate tokenizer: BertTokenizer for BERT and DistilBertTokenizer for DistilBERT. These tokenizers perform subword tokenization, splitting words into subword units based on a fixed vocabulary. This ensures that even rare or misspelled words can be represented effectively through shared subword embeddings.
The tokenization process outputs the following components for each tweet:

- Input IDs: Sequences of integers representing tokens/subwords, mapped to entries in the model's vocabulary.

- Attention Masks: Binary masks indicating which tokens are actual content (1) and which are padding (0). This ensures the model does not attend to padded positions during training.

To maintain consistent input shapes for batching and GPU efficiency, all tokenized sequences were padded or truncated to a fixed maximum length (max_length = 60),

based on prior analysis of tweet lengths. The resulting input IDs and attention masks are converted into PyTorch tensors and wrapped into TensorDataset objects. These are then loaded into DataLoader instances for efficient batching during training and evaluation.
This vectorization strategy leverages the full capabilities of pretrained language models, preserving both the syntactic and semantic structure of the tweets while ensuring compatibility with the deep learning pipeline.

3. Algorithms and Experiments

3.1. Experiments

3.1.1. BERT.

1. Baseline Experiment:
   As an initial step, I implemented a baseline BERT model in order to establish a reference point for evaluating future improvements. The architecture was intentionally kept simple, using default hyperparameters: a batch size of 16, a learning rate of 2e-5, 2 training epochs, the Adam optimizer, a linear learning rate scheduler, and the default dropout rate of 0.1.
   This configuration served a dual purpose: it allowed me to gain familiarity with the end-to-end BERT training process in PyTorch, and it provided a reliable performance benchmark for subsequent experimentation. The baseline model achieved a validation accuracy of approximately 85.24%, highlighting the strong capabilities of BERT even with minimal tuning. Importantly, this experiment also underscored the potential for further improvement through architectural adjustments, fine-tuning strategies, and regularization techniques.

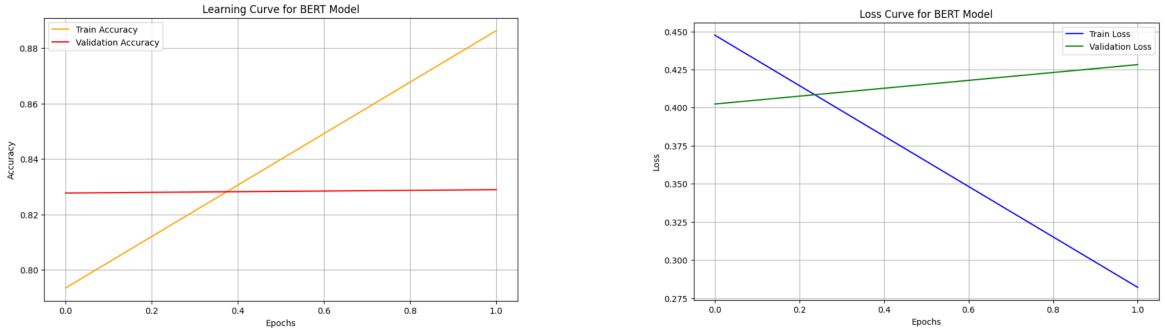

Figure 5: Baseline BERT model

2. Testing Learning Rates
   An essential part of the experimentation involved assessing the impact of different learning rates on model performance. Starting from the default setting of 2e-5, I conducted experiments by both decreasing and increasing the learning rate to observe its effect on convergence and accuracy. When the learning rate was reduced to 1e-5, the model exhibited slower convergence and underwhelming

validation accuracy, 85.17%. The smaller step size limited the optimizer's ability to escape local minimum and required more epochs to achieve competitive results. Conversely, increasing the learning rate to 3e-5 led to unstable updates and poorer generalization. The model appeared to overshoot optimal minimum, resulting in higher validation loss and lower accuracy, specifically 85.14% due to insufficient fine-grained optimization.
Through empirical evaluation, 2e-5 was confirmed as the most effective learning rate for this setup, achieving a validation accuracy of 85.24%. This balance between stability and convergence speed made it the optimal choice for subsequent training runs.

3. Varying the Batch Size
   Another key experiment aimed at improving model performance involved tuning the batch size. The batch size directly affects the stability and speed of gradient updates, as well as the generalization ability of the model. Starting with the default value of 16, I increased the batch size to 32, which led to a noticeable improvement in validation accuracy, reaching 85.33%. This improvement is attributed to more stable gradient estimates and better GPU utilization, which allowed for more efficient training without compromising generalization. However, further increasing the batch size to 64 resulted in a decrease in validation accuracy. While larger batches can accelerate training by reducing the number of updates per epoch, they often lead to poorer generalization due to reduced gradient noise, which plays a regularizing role in training deep models.
   Based on these results, a batch size of 32 was selected as the optimal trade-off between training efficiency and model performance. It provided the best validation accuracy without introducing overfitting or underfitting effects. These findings are consistent with the broader deep learning literature, where moderately sized batches often yield better generalization than very large ones, especially in transformer-based architectures.

4. Increasing Number of Training Epochs:
   To explore whether additional training could enhance the model's performance, I extended the number of training epochs from the baseline of 2 to 3 and 4. The rationale behind this adjustment was to provide the model with more opportunities to learn complex patterns from the data and improve its generalization capabilities. However, prolonged training poses the risk of overfitting, where the model begins to memorize training examples rather than learning generalizable features. To counter this, I implemented early stopping, a regularization technique that monitors the validation loss and terminates training if no improvement is observed for a specified number of consecutive epochs (patience). Early stopping offers a practical balance between underfitting (due to insufficient training) and overfitting (due to excessive training). By halting the learning process at the point of optimal validation performance, it helps preserve model generalization to unseen data. Despite this adjustment, the final validation accuracy slightly decreased to 85.19%, suggesting that additional epochs did not necessarily benefit the model. This result highlights that extending training time alone does not guarantee better performance and reinforces the importance of carefully monitoring validation metrics during training.

5. Optimizer Selection
   As training progressed, the choice of optimizer emerged as a critical hyperparameter for enhancing the performance and stability of the BERT model. The default configuration employs the Adam optimizer, which is known for its effectiveness in transformer-based architectures due to its adaptive learning rate capabilities. To explore potential improvements, I experimented with the AdamW optimizer, an extension of Adam that decouples weight decay from the gradient update step. This distinction is particularly beneficial for models like BERT, where applying weight decay correctly can improve generalization without interfering with the optimization dynamics. In these experiments, AdamW was applied with a weight decay factor of 0.01. The results confirmed the expected benefits: validation accuracy improved compared to the standard Adam optimizer, specifically it increased to 85.38%. The regularization effect introduced by weight decay helped prevent overfitting, especially as the number of epochs increased, and led to more stable convergence across training runs.

6. Experiments on Learning Rate Schedulers
   While the default linear learning rate scheduler already contributed to stable training and competitive accuracy, further experimentation with alternative scheduling strategies was conducted to explore potential improvements in model generalization and convergence dynamics. Specifically, I evaluated the performance of two additional schedulers: the cosine annealing scheduler, which gradually decreases the learning rate following a cosine curve without abrupt changes, and the polynomial decay scheduler, which reduces the learning rate according to a predefined polynomial function over the course of training. Despite offering smoother decay dynamics, neither scheduler outperformed the linear baseline in terms of validation accuracy. These results suggest that while alternative schedulers may affect training behavior, the linear scheduler remains the most effective choice for this task and dataset.

7. Dropout Regularization
   As a final step in hyperparameter tuning, I examined the impact of dropout, a widely used regularization technique that helps prevent overfitting by randomly deactivating neurons during training. The default dropout rate of 0.1 in the BERT model is applied to both the attention layers and the feedforward layers. To explore whether stronger regularization could enhance generalization, I increased the dropout rate to 0.2, aiming to reduce the model's reliance on specific neuron activations. However, this more aggressive strategy did not lead to any noticeable improvement in validation accuracy. In some runs, it even caused minor performance degradation, likely due to excessive regularization interfering with learning dynamics.
   As a result, the default dropout rate of 0.1 was retained, as it offered the best balance between regularization and model capacity, allowing BERT to generalize well without hindering its ability to learn meaningful patterns from the data.

3.1.2. Table of trials. The table below summarizes the key experiments conducted during model development, highlighting the impact of each architectural or training change on validation accuracy. These trials reflect a step-by-step approach to incrementally improve the model's performance through informed experimentation:

Trial	Experiment	Score
0	Baseline BERT model (tutorial's default settings)	85.24%
1	Learning rate testing (1e-5, 3e-5); best: 2e-5	85.24%
2	Increased batch size to 32	85.33%
3	Increased number of training epochs to 4	85.19%
4	Switched optimizer to AdamW with weight decay 0.01	85.38%
5	Replaced linear scheduler with cosine scheduler	85.28%
6	Increased dropout rate to 0.2	85.16%
7	Final BERT model	85.38%

Table 1: Trials on BERT model

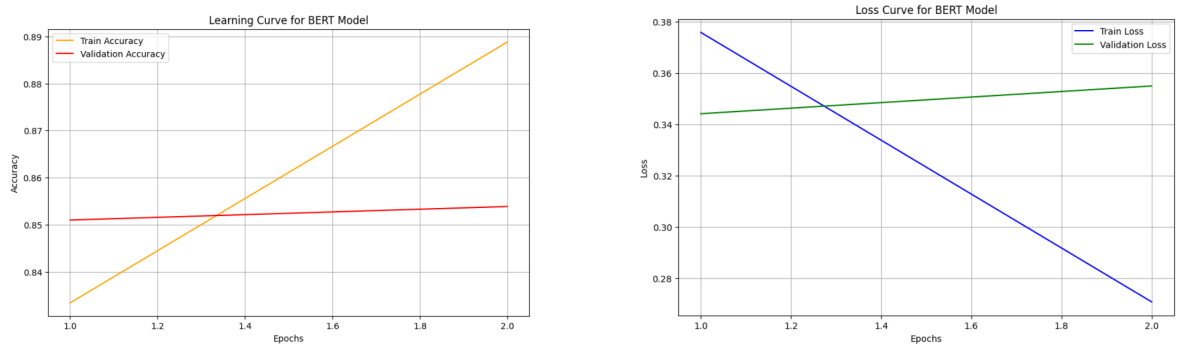

Figure 6: Final BERT model

3.1.3. DistilBERT.

1. Baseline Experiment – DistilBERT
   As an initial step, I implemented a baseline DistilBERT model to establish a reference point for future improvements. The model was trained using default hyperparameters, batch size 16, learning rate 2e-5, 2 epochs, the Adam optimizer, a linear learning rate scheduler, and the default dropout rate of 0.1, mirroring the initial BERT setup.
   This baseline served two key purposes: it provided hands-on experience with DistilBERT training in PyTorch and established a reliable benchmark. The model achieved a validation accuracy of 84.73%, demonstrating the strong performance of DistilBERT even with minimal tuning. Moreover, this experiment highlighted opportunities for further improvement through hyperparameter optimization and regularization strategies, which guided the design of subsequent experiments.

2. Testing Learning Rates
   To evaluate the impact of learning rate on DistilBERT model performance, I tested three values: 1e-5, 2e-5 (default), and 3e-5. Reducing the rate to 1e-5 resulted in slower convergence and slightly lower validation accuracy (84.68%), while increasing it to 3e-5 led to unstable training and degraded accuracy (84.64%) due to overshooting and poor generalization. The default value of 2e-5 yielded the best result (84.73%), offering a stable and efficient learning process. These results confirm the importance of tuning the learning rate, even within standard ranges, as transformer models like DistilBERT are highly sensitive to this parameter.

3. Testing Batch Sizes

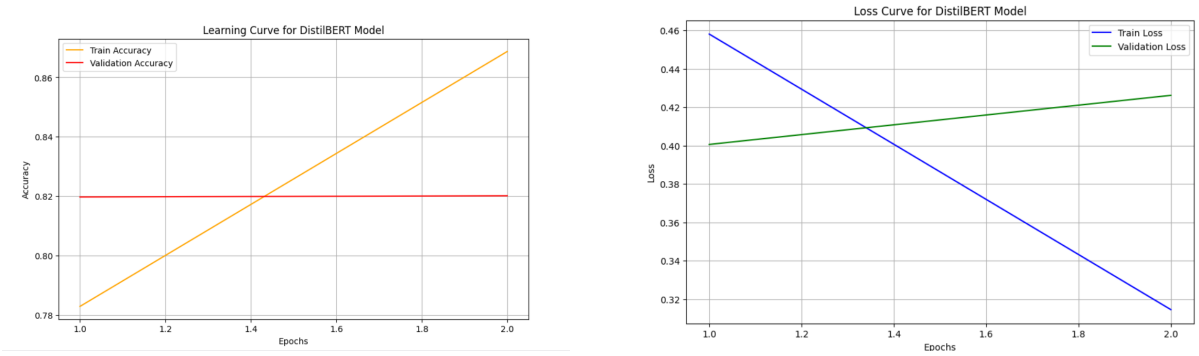

Figure 7: Baseline DistilBERT model

TTo improve training efficiency and performance, I tested three batch sizes: 16, 32, and 64. Increasing the batch size from the default 16 to 32 resulted in a drop in validation accuracy to 84.68%, despite improved GPU utilization and training speed. Further increasing it to 64 led to a slightly lower accuracy of 84.62%, likely due to reduced gradient noise and weaker generalization. Based on these results, batch size 16 was selected as the optimal configuration.

4. Increasing Number of Training Epochs:
   To assess the impact of extended training, I increased the number of epochs for DistilBERT from 2 to 3 and 4. While the goal was to give the model more capacity to learn complex patterns, longer training led to a slight drop in validation accuracy (84.63%), likely due to overfitting. To mitigate this, early stopping was employed, halting training when no improvement in validation loss was observed for several epochs. These findings suggest that for DistilBERT, additional training time does not necessarily yield better performance, and careful monitoring is essential to avoid overfitting, so the number of epochs was defined in 2.

5. Optimizer Selection
   During experimentation, the choice of optimizer proved crucial for BERT's performance and training stability. While the default Adam optimizer is widely used due to its adaptive learning rate, I explored the AdamW variant, which decouples weight decay from gradient updates. This adjustment allows for more effective regularization without disrupting optimization dynamics. Using AdamW with a weight decay of 0.01 led to a validation accuracy improvement to 84.78%, outperforming standard Adam. The added regularization helped prevent overfitting and stabilized convergence, especially as training epochs increased. Based on these results, AdamW was selected as the final optimizer for BERT, offering a better balance between generalization and efficiency.

6. Experiments on Learning Rate Schedulers
   To evaluate the effect of learning rate scheduling on DistilBERT's performance, I compared the default linear scheduler with cosine annealing and polynomial decay alternatives. While both alternatives offered smoother decay curves, they did not improve validation accuracy compared to the linear baseline, specifically decreased model's accuracy to 84.69%. The linear scheduler remained the most effective, offering a stable balance between convergence and generalization. These

results suggest that for DistilBERT, more complex scheduling strategies do not necessarily yield performance gains.

7. Dropout Regularization
   To examine the effect of regularization on generalization, I tested an increased dropout rate of 0.2, compared to the default 0.1 used in DistilBERT. While the higher dropout introduced stronger noise during training, it did not lead to improved validation accuracy. In fact, the model performed best with the default 0.1, suggesting that the lighter regularization was more suitable for DistilBERT's compact architecture. As a result, dropout = 0.1 was retained for the final configuration.

3.1.4. Table of trials. The table below summarizes the key experiments conducted during model development, highlighting the impact of each architectural or training change on validation accuracy. These trials reflect a step-by-step approach to incrementally improve the model's performance through informed experimentation:

Trial	Experiment	Score
0	Baseline BERT model (tutorial's default settings)	84.73%
1	Learning rate testing (1e-5, 3e-5); best: 2e-5	84.73%
2	Increased batch size to 32	84.68%
3	Increased number of training epochs to 4	84.63%
4	Switched optimizer to AdamW with weight decay 0.01	84.78%
5	Replaced linear scheduler with cosine scheduler	84.69%
6	Increased dropout rate to 0.2	84.66%
6	Final DistilBERT model	84.78%

Table 2: Trials on DistilBERT model

3.2. Hyper-parameter tuning

3.2.1. BERT. After all, this comprehensive training phase yielded strong and consistent performance, achieving 85.65% accuracy on the validation set and 85.63% accuracy on the test set. These results confirm that the model generalizes well across both seen and unseen data.
For this configuration, the BERT tokenizer was employed to convert tweets into a format compatible with the transformer architecture. Before tokenization, an extensive text preprocessing pipeline was applied to clean and normalize the raw tweets. This involved removing user mentions, URLs, and noisy artifacts typical of informal language on social media, as well as correcting contractions and common misspellings. This preprocessing ensured that the tokenized inputs contained high-quality linguistic features, allowing the pretrained BERT embeddings to capture nuanced semantic patterns relevant to sentiment.

After systematic experimentation and tuning, the optimal configuration for the BERT model was determined as follows:

- Learning Rate = 2e-5:
  This value offered the best balance between convergence speed and training stability. It was high enough to allow efficient learning within a limited number of epochs, yet low enough to prevent overshooting or divergence during training.

- Batch Size = 32:
  Empirical results showed that a batch size of 32 yielded the highest validation accuracy. It provided a good compromise between computational efficiency and model generalization, producing smoother gradient estimates and making effective use of available GPU resources.

- Number of Epochs = 2:
  Although additional epochs were tested (e.g., 3 and 4), training for 2 epochs was found to be the most effective when combined with early stopping and other regularization techniques. Longer training tended to result in diminishing returns or slight overfitting.

- AdamW Optimizer with weight decay = 0.01:
  Among the tested optimizers, AdamW offered the best performance. It adapts learning rates for each parameter based on estimates of first and second moments of the gradients, enabling fast and stable convergence.

- Linear Scheduler:
  While alternatives such as cosine annealing and polynomial decay were tested, the linear learning rate scheduler consistently yielded better performance. Its gradual and predictable decay provided stable convergence without introducing unnecessary training noise.

- Dropout = 0.1:
  The default dropout rate of 0.1 proved optimal for regularizing the BERT model. Increasing it to 0.2 led to decreased validation accuracy, indicating that stronger regularization was unnecessary. Dropout = 0.1 offered an effective balance, preventing overfitting while preserving learning capacity.

Underfitting or Overfitting?

- Underfitting:
  Underfitting occurs when a model is unable to capture the underlying patterns of the training data, typically resulting in low accuracy on both training and validation sets. In this case, no signs of underfitting are observed. The training accuracy increases steadily, reaching approximately 89%, and the training loss decreases consistently across epochs. These trends indicate that the model is effectively learning from the training data and has sufficient capacity to fit it well.

- Overfitting:
  Overfitting is indicated when the model performs significantly better on the training data than on the validation data, suggesting it has memorized training examples rather than learning generalizable patterns. In the BERT training curves, a clear gap emerges: while training accuracy continues to increase, validation accuracy shows only marginal improvement, plateauing around 85.6%. Additionally, the validation loss increases slightly during the second epoch, while the training

loss continues to decline. This divergence between training and validation performance suggests the model begins to overfit after the first epoch. Although the overfitting is not severe, it justifies the use of early stopping and regularization techniques such as dropout and weight decay to control generalization error.

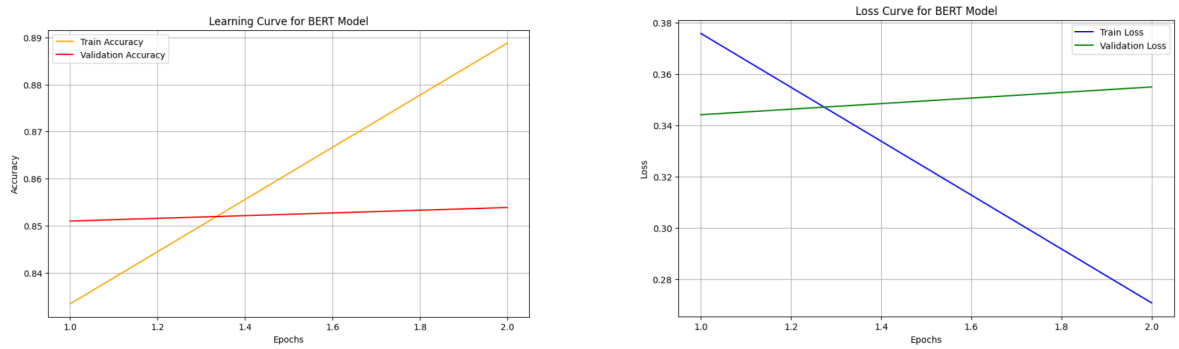

Figure 8: Final BERT model

3.2.2. DistilBERT. The training phase yielded strong and consistent performance, achieving 84.78% accuracy on the validation set and 85.79% accuracy on the test set. These results confirm that the model generalizes well across both seen and unseen data.
For this configuration, the DistilBERT tokenizer was employed to convert tweets into a format compatible with the transformer architecture. Before tokenization, an extensive text preprocessing pipeline was applied to clean and normalize the raw tweets. This involved removing user mentions, URLs, and noisy artifacts typical of informal language on social media, as well as correcting contractions and common misspellings. This preprocessing ensured that the tokenized inputs contained high-quality linguistic features, allowing the pretrained BERT embeddings to capture nuanced semantic patterns relevant to sentiment.

After extensive experimentation, the optimal configuration for the DistilBERT model was identified as follows:

- Learning Rate = 2e-5:
  This value offered the most effective trade-off between convergence speed and training stability. It allowed the model to learn efficiently within a small number of epochs without causing training instability or divergence.

- Batch Size = 16:
  Among the tested values (16, 32, 64), a batch size of 16 consistently delivered the best validation accuracy. Smaller batches improved generalization, likely due to increased gradient noise acting as an implicit regularizer, which is particularly beneficial for compact models like DistilBERT.

- Number of Epochs = 2:
  Although additional epochs were tested (e.g., 3 and 4), training for 2 epochs was found to be the most effective when combined with early stopping and other regularization techniques. Longer training tended to result in diminishing returns or slight overfitting.

- AdamW Optimizer with weight decay = 0.01:
  Among the tested optimizers, AdamW offered the best performance. It adapts learning rates for each parameter based on estimates of first and second moments of the gradients, enabling fast and stable convergence.

- Linear Scheduler:
  Alternative schedulers such as cosine annealing and polynomial decay were tested but did not outperform the linear scheduler. The linear schedule provided smooth, predictable learning rate decay that supported stable and effective convergence.

- Dropout = 0.1:
  The default dropout rate of 0.1 proved optimal for regularizing the BERT model. Increasing it to 0.2 led to decreased validation accuracy, indicating that stronger regularization was unnecessary. Dropout = 0.1 offered an effective balance, preventing overfitting while preserving learning capacity.

Underfitting or Overfitting

- Underfitting:
  Underfitting occurs when the model fails to learn from the training data, resulting in low accuracy and high loss in both training and validation. In this case, DistilBERT shows no signs of underfitting: training accuracy steadily increases (up to 88%) and training loss consistently decreases, indicating effective learning and sufficient model capacity.

- Overfitting:
  Overfitting is evident when a model performs well on training data but poorly on validation data. In DistilBERT, training accuracy rises sharply, while validation accuracy improves only slightly (84.6%) and validation loss increases, indicating early signs of overfitting. Although mild, this highlights the importance of techniques like dropout, early stopping, and proper optimizer/scheduler selection to preserve generalization.

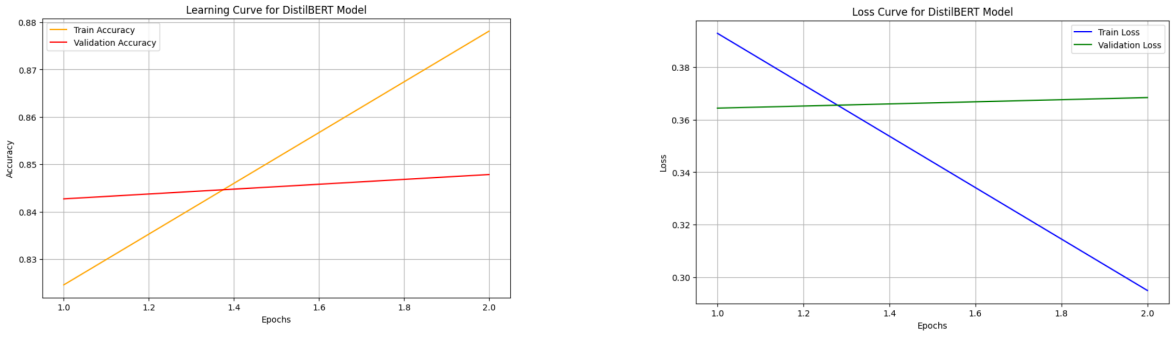

Figure 9: Final DistilBERT model

3.3. Optimization techniques

To enhance the performance of the BERT and DistilBERT model, a combination of manual experimentation and automated hyperparameter optimization techniques was

employed. These methods were applied iteratively throughout models development, allowing for informed decisions grounded in empirical results.

Manual Experiments
Extensive manual tuning was carried out across several components of the models. Each of these contributed to incremental performance improvements:

- Text Preprocessing Function (primarily in Exercise 1):
  The preprocessing function was refined to handle noisy Twitter data effectively. This included steps such as lowercasing, tokenization, contraction expansion, removal of mentions, hashtags, URLs, and correction of common spelling errors. These improvements helped the BERT Tokenizer better reflect the sentiment-carrying structure of each tweet.

- Training Hyperparameters of BERT model:
  Systematic experimentation was conducted across key training hyperparameters to optimize BERT's performance. Learning rates ranging from 1e-5 to 3e-5 were evaluated, with 2e-5 offering the best balance between stability and convergence. Among batch sizes 16, 32, and 64, a size of 32 achieved the highest accuracy while maintaining generalization. Although longer training was tested, 2 epochs combined with early stopping proved optimal, avoiding overfitting. The AdamW optimizer with weight decay 0.01 outperformed standard Adam in terms of generalization. Finally, among tested schedulers, linear, cosine annealing, and polynomial decay, the cosine scheduler delivered the most consistent improvements in validation accuracy.

- Training Hyperparameters of DistilBERT model:
  To optimize DistilBERT's performance, I systematically tuned key training hyperparameters. I tested learning rates between 1e-5 and 3e-5, with 2e-5 yielding the best results in terms of convergence and stability. For batch size, 16 proved most effective, outperforming 32 and 64 in validation accuracy and generalization. Although training for more epochs was explored, 2 epochs with early stopping offered the best trade-off, preventing overfitting. The AdamW optimizer with a weight decay of 0.01 led to better generalization compared to standard Adam. Lastly, among all learning rate schedulers tested, the cosine scheduler provided the most consistent gains in validation performance.

These manual experiments were guided by empirical observation of training curves, validation scores, and generalization behavior. They were essential for building intuition about model sensitivity and tuning priorities.

Automated Optimization with Optuna
In addition to manual trials, I employed the Optuna optimization framework to automatically search for the most effective hyperparameters. Optuna uses a sampling-based approach to efficiently explore the hyperparameter space and identify high-performing configurations. Specifically, the automated optimization process focused on tuning three key parameters:

- Dropout

- Learning rate

- Choice of optimizer (Adam vs. AdamW)

Although running Optuna is computationally intensive (each run takes approximately 3 hours), it provided valuable insights that complemented the manual trials. For this reason, the corresponding code was commented out in the final submission for practicality, but it remains available and can be activated for further evaluation.
The results obtained through Optuna were consistent with those from manual experimentation for both BERT and DistilBERT model, effectively validating the selected training configuration and confirming the robustness of the final model architecture.

3.4. Evaluation

To assess the effectiveness of the 2 models, BERT and DistilBERT, I evaluated its predictions using accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of the model's performance beyond just accuracy as it was showed in tutorial lesson.

Specifically:

- Accuracy: Measures the overall correctness of the model by calculating the ratio of correctly classified instances to the total instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- Precision: Evaluates how many of the predicted positive instances were actually correct, helping to measure false positives.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

- Recall : Indicates how many of the actual positive instances were correctly identified, focusing on false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

- F1-score: The harmonic mean of precision and recall, providing a balanced measure.

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

Perfomance Results for BERT model:
The evaluation results on the validation dataset are shown in the table below:

The final BERT model demonstrates strong and balanced performance across both sentiment classes. Precision and recall are consistently high for both Class 0 (negative) and Class 1 (positive), with F1-scores of 0.86 and 0.85, respectively. The overall accuracy is 85%, indicating reliable generalization. Moreover, the macro and weighted averages confirm that the model performs equally well across both classes, with no signs of class imbalance or prediction bias. These results validate the effectiveness of the chosen architecture and hyperparameters.

	Precision	Recall	F1-score	Support
Class 0	0.85	0.86	0.86	21197
Class 1	0.86	0.84	0.85	21199
Accuracy			0.85	42396
Macro avg	0.85	0.85	0.85	42396
Weighted avg	0.85	0.85	0.85	42396

Table 3: Evaluation Metrics for the Final BERT Model

	Precision	Recall	F1-score	Support
Class 0	0.85	0.85	0.85	21197
Class 1	0.85	0.85	0.85	21199
Accuracy			0.85	42396
Macro avg	0.85	0.85	0.85	42396
Weighted avg	0.85	0.85	0.85	42396

Table 4: Evaluation Metrics for the Final DistilBERT Model

Perfomance Results for DistilBERT model:
The evaluation results on the validation dataset are shown in the table below: The DistilBERT model achieves high and balanced performance across both sentiment classes. Precision, recall, and F1-score are all 0.85 for Class 0 and Class 1, indicating that the model classifies both sentiments with equal effectiveness. The overall accuracy is also 85%, matching the performance of the full BERT model despite DistilBERT's smaller size and lower complexity. The macro and weighted averages further confirm the model's consistency and fairness, with no indication of class imbalance. These results highlight DistilBERT's efficiency in delivering competitive performance while being computationally lighter.

3.4.1. ROC curve. Both BERT and DistilBERT models exhibit excellent classification performance, as shown by their ROC curves. The AUC scores of 0.9319 (BERT) and 0.9256 (DistilBERT) indicate high discriminative power, meaning the models are very effective at distinguishing between positive and negative sentiment. The curves lie well above the diagonal baseline, confirming that both models outperform random guessing by a wide margin. Although BERT has a slightly higher AUC, both models demonstrate strong and reliable generalization.

3.4.2. Learning Curve. Both BERT and DistilBERT models exhibit steady improvements in training accuracy and consistent reductions in training loss over the course of training. At the same time, validation accuracy and loss remain relatively stable, without drastic fluctuations. This behavior indicates effective learning and convergence, while also revealing mild signs of overfitting, as a small performance gap between training and validation begins to emerge. These

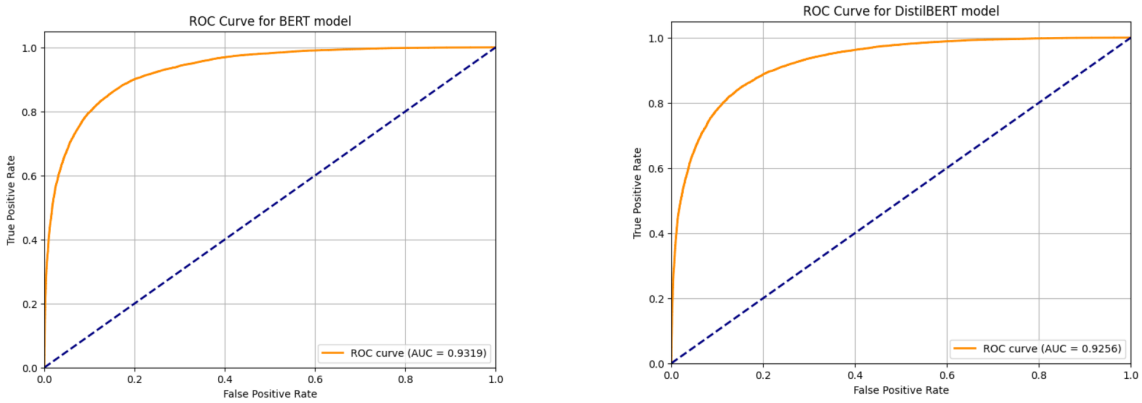

Figure 10: ROC Curves in BERT and DistilBERT models

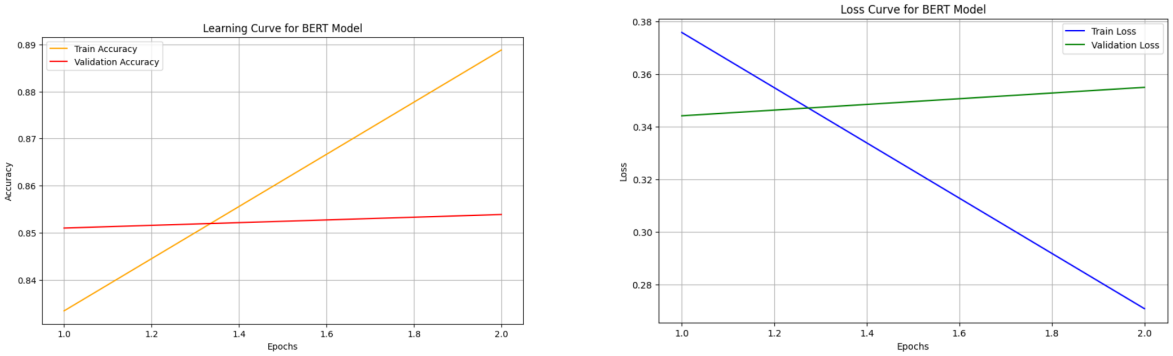

Figure 11: Learning Curves in BERT model

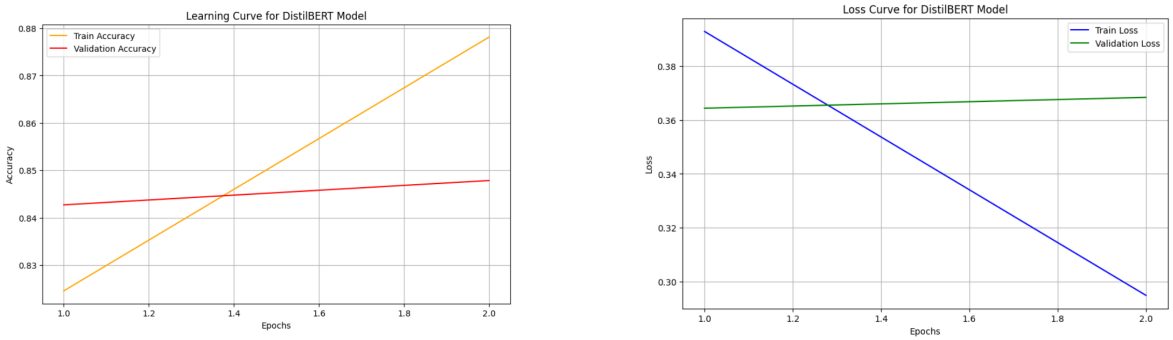

Figure 12: Learning Curves in DistilBERT model

3.4.3. Confusion matrix. Both BERT and DistilBERT achieve balanced classification performance, with high true positive and true negative counts. BERT shows slightly fewer false positives and false negatives than DistilBERT, confirming its marginally better accuracy. The confusion matrices indicate no class bias, and both models generalize well across sentiment labels.

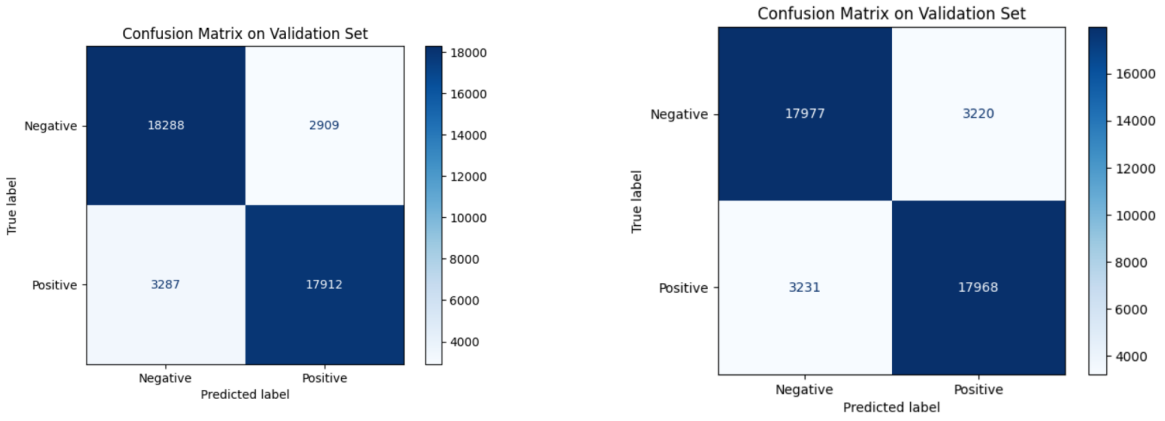

Figure 13: Confusion Matrices in BERT and DistilBERT models

4. Results and Overall Analysis

4.1. Results Analysis

The final BERT model achieved a validation accuracy of 85.38% and a test accuracy of 85.66% (on Kaggle), while DistilBERT followed closely with 84.78% on the validation set and 84.79% on the test set. These results demonstrate strong and consistent generalization across both seen and unseen data, confirming the effectiveness of fine-tuned transformer-based models in handling noisy and informal social media text such as tweets.
Specifically,
Evaluation Metrics on Validation Set of BERT model

- Accuracy: 0.85385
- Precision: 0.86029
- Recall: 0.84495
- F1-Score: 0.85255

And,
Evaluation Metrics on Validation Set of DistilBERT model

- Accuracy: 0.84784
- Precision: 0.84803
- Recall: 0.84759

- F1-Score: 0.84781

These metrics show that BERT performs slightly better overall, especially in precision and F1-score, though DistilBERT remains highly competitive despite its reduced size and training time. In addition, the ROC curves (see Section 3.4) confirm the models' strong discriminative ability. BERT achieved an AUC of 0.9319 and DistilBERT 0.9256, both significantly above random guessing and indicative of robust performance in class separation. The learning and loss curves (see Section 3.4) reveal stable and efficient training for both models. Training accuracy increases steadily while training loss decreases, and validation accuracy remains stable, suggesting effective learning with minor overfitting. The use of early stopping, dropout, and weight decay played a key role in maintaining generalization. Finally, the confusion matrices (see Section 3.4) show well-balanced performance across both sentiment classes. BERT made slightly fewer false predictions, reinforcing its edge in precision. Both models maintained class balance, with no indication of bias toward positive or negative sentiment.
In summary, the results clearly exceed initial expectations and indicated that this is a good performance. Achieving over 85% accuracy and F1-score on a challenging, informal dataset like Twitter demonstrates the effectiveness of transformer-based models such as BERT and DistilBERT. This strong performance reflects not only the power of pretrained contextual embeddings, but also the impact of careful preprocessing, hyperparameter tuning, and regularization techniques.

4.1.1. Best trial. The best-performing configurations for both BERT and DistilBERT shared a common training pipeline with carefully tuned components and consistent preprocessing:

- Preprocessing:
  Extensive cleaning and normalization, including lowercasing, tokenization, contraction handling, removal of noise (mentions, hashtags, URLs), and spelling correction.
- Tokenizer:
  HuggingFace's pretrained BertTokenizer and DistilBertTokenizer were used to convert text into subword token IDs compatible with each model.
- Architecture:
  Transformer-based fine-tuned versions of BERT and DistilBERT, initialized from pretrained checkpoints (bert-base-uncased and distilbert-base-uncased).
- Training Hyperparameters
  - Optimizer: AdamW with weight decay = 0.01
  - Learning Rate: 2e-5
  - Batch Size: 32 for BERT, 16 for DistilBERT (based on memory and performance trade-offs)
  - Epochs: 2, combined with early stopping
  - Dropout Rate: 0.1
  - Learning Rate Scheduler: Linear

4.2. Comparison with the first project

In the first project, a Logistic Regression model with TF-IDF features was used for sentiment classification, achieving a final validation accuracy of 80.% and an F1-score of 0.80. This performance is strong considering the model's simplicity and interpretability. However, both BERT and DistilBERT, fine-tuned in this project, significantly outperformed the classical approach. The BERT model achieved a validation accuracy of 85.6% and an F1-score of 0.85, demonstrating a notable performance boost. Similarly, DistilBERT, a lighter and faster variant, reached a validation accuracy of 85%, matching BERT's performance closely while requiring fewer computational resources. Both transformer-based models benefit from contextualized word embeddings, pretraining on massive corpora, and the ability to capture deep semantic and syntactic patterns, which traditional models like Logistic Regression cannot. While the first project relied on sparse frequency-based representations that treat words independently, the transformer models understand word meaning in context and account for word order and dependencies, which is especially valuable in informal and noisy text such as tweets. That's why both BERT and DistilBERT show clear advantages in performance and linguistic understanding.

4.3. Comparison with the second project

In the second project, a Deep Neural Network (DNN) with GloVe Twitter embeddings was developed for sentiment classification, achieving a validation accuracy of 79.09% and an F1-score of 0.789. While this was a notable improvement over simpler baselines, both BERT and DistilBERT in the current project achieved higher performance, with BERT reaching 85.6% accuracy and DistilBERT 85%, each with F1-scores of 0.85. The key difference lies in the contextual understanding of language. Unlike GloVe embeddings, which are static and averaged, BERT-based models dynamically adapt word representations based on context. This makes them more effective at capturing sentiment nuances, especially in noisy text like tweets. Additionally, BERT required less manual tuning and fewer architectural decisions, yet consistently delivered better generalization. Despite being more computationally demanding, transformer-based models offer superior accuracy and linguistic depth, making them more suitable for real-world sentiment analysis tasks.

5. Bibliography

References

[1] HuggingFace. https://huggingface.co/docs/transformers/en/main_classes/tokenizer.

[2] Manolis Koubarakis. Bert. 2025.

[3] Manolis Koubarakis. Machine translation and attention. 2025.

[4] Manolis Koubarakis. Transformers. 2025.

[3] [4] [2] [1]