UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Andreakis Dimitrios*
sdi2300008

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

Contents

1. Abstract

Sentiment analysis is a crucial task in Natural Language Processing (NLP), enabling the classification of text into positive or negative sentiments. In this project, we aim to develop a sentiment classification model using transformer-based architectures, specifically BERT and DistilBERT. The dataset consists of labeled tweets, which are preprocessed and tokenized using pretrained tokenizers. We utilize exploratory data analysis (EDA) to understand the dataset's structure and characteristics. Model performance is evaluated using various metrics, and hyperparameter tuning is performed using Optuna to improve results. DistilBERT is also considered as a lighter, faster alternative to BERT with competitive accuracy.

2. Data processing and analysis

2.1. Pre-processing

Unlike traditional machine learning pipelines where extensive text cleaning is often necessary, transformer-based models such as BERT and DistilBERT are designed to operate effectively on raw text. Therefore, minimal pre-processing was required in this project.

We utilized the pretrained tokenizer corresponding to each model (`BertTokenizer` for BERT and DistilBERT), which automatically handles lowercasing, punctuation splitting, and the addition of special tokens such as `[CLS]` and `[SEP]`. The tokenizer also performs truncation and padding to a fixed maximum length (`max_len`), which ensures uniform input sizes for batching.

Each tweet was tokenized using `encode_plus`, which returns the input IDs and attention masks needed for the model. These tokenized inputs were stored in a custom `Dataset` class for efficient loading during training. Since the models are robust to noisy or informal language, no additional steps such as removing URLs, mentions, or emojis were applied.

2.2. Analysis

To better understand the dataset, we conducted an Exploratory Data Analysis (EDA), focusing on dataset characteristics, class distribution, text length, and common word patterns.

2.2.1. Dataset Overview. The dataset consists of three partitions: training, validation, and test sets. Their shapes are as follows:

- **Training set:** 148,388 samples, 3 columns
- **Validation set:** 42,396 samples, 3 columns
- **Test set:** 21,199 samples, 2 columns

We checked for missing values and found none. Additionally, there were no duplicate tweets.

2.2.2. Class Distribution. To ensure a balanced dataset, we examined the class distribution. The training set consists of two labels:

- **Label 0 (Negative Sentiment)**
- **Label 1 (Positive Sentiment)**

A visualization of the class distribution is provided below:

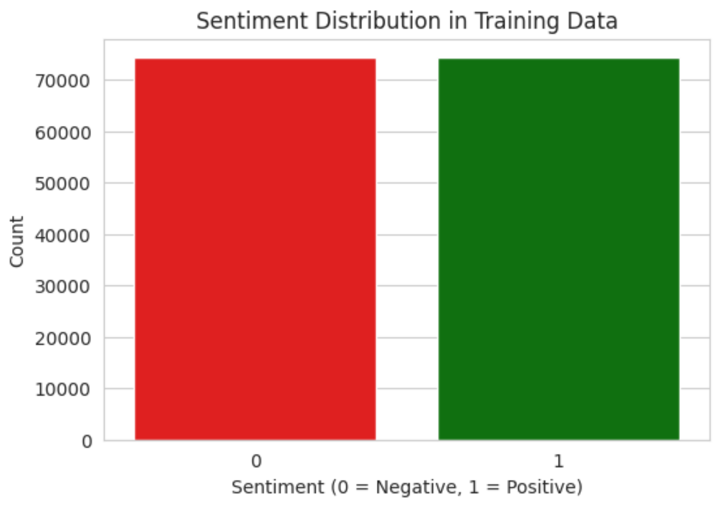

Figure 1: Class distribution of sentiments in the training set.

2.2.3. Text Length Distribution. To assess variability in tweet length, we calculated the number of words per tweet. The distribution is shown below:

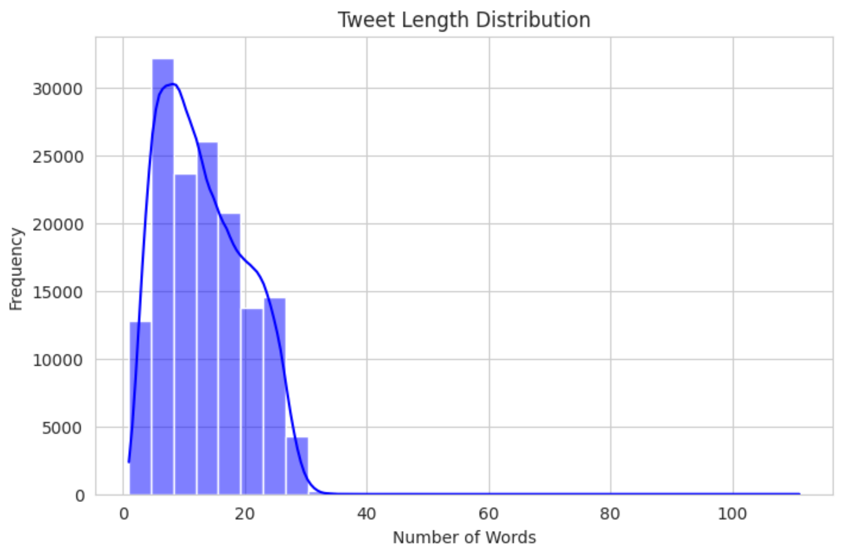

Figure 2: Histogram of tweet length distribution.

2.2.4. Word Cloud Analysis. We generated word clouds to visualize the most common words in positive and negative tweets.

- **Positive Sentiment:** Frequently occurring words in positive tweets.
- **Negative Sentiment:** Frequently occurring words in negative tweets.

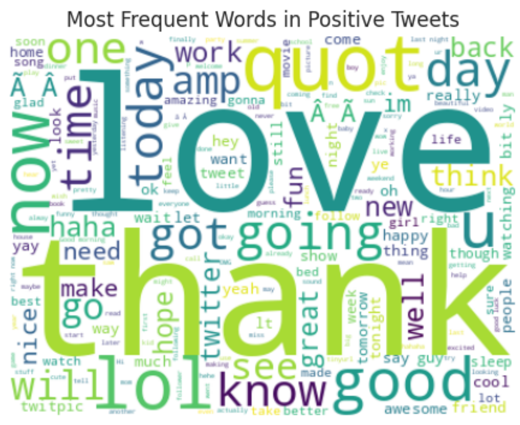

Figure 3: Word cloud of positive sentiment tweets.

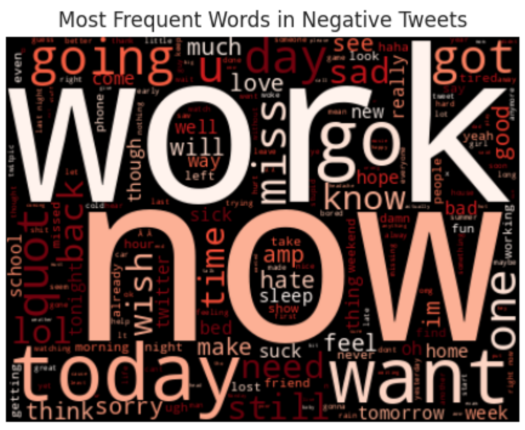

Figure 4: Word cloud of negative sentiment tweets.

2.2.5. Key Observations.

- The dataset is balanced in terms of sentiment distribution.
- Tweets vary in length, with most tweets containing around 6 words.
- Positive tweets commonly include words like "thank" or "love", while negative tweets feature "work" or "now".

2.3. Data partitioning for train, test and validation

The dataset was provided in three pre-defined partitions:

- **Training set:** 148,388 samples
- **Validation set:** 42,396 samples
- **Test set:** 21,199 samples

Since the dataset was already partitioned, we used these splits directly without further modification.

2.4. Vectorization

In this project, we do not employ traditional vectorization methods such as TF-IDF. Instead, we leverage the power of transformer-based models, which incorporate a built-in mechanism for converting raw text into dense, contextualized vector representations.

Specifically, the input tweets are first tokenized using the appropriate tokenizer for each model (BERT or DistilBERT). The tokenizer maps each word or subword token to a corresponding integer ID from the model's vocabulary. These token IDs, along with attention masks, are then passed to the model, which produces embeddings through its multiple self-attention layers.

The output embeddings capture rich semantic and syntactic information based on the context of each word within the sentence. These contextualized representations are significantly more expressive and informative compared to traditional static embeddings, making them well-suited for downstream tasks such as sentiment classification.

3. Algorithms and Experiments

3.1. Experiments

We approached the sentiment classification task incrementally. Initially, we started with a brute-force BERT configuration using common hyperparameter choices. As we observed overfitting and unstable validation loss, we iterated over several trials adjusting parameters such as learning rate, batch size, number of epochs, and regularization through weight decay and dropout.

We later automated hyperparameter tuning using `Optuna`, which helped us find a more balanced configuration that minimized validation loss and improved generalization. In addition to the BERT model, we also tested `DistilBERT`, a lighter transformer model, using the best hyperparameters found via Optuna.

BERT Experiments:

- In the first brute-force attempt, we trained BERT for 4 epochs with a learning rate of 3×10^{-5} and batch size of 32. Although performance improved, the validation loss started to increase after epoch 3, indicating overfitting.

- The second attempt reduced the number of epochs to 3 and added weight decay (0.02). Validation loss stabilized better, but overfitting still occurred.

- The third and best-performing BERT trial was selected via Optuna. It used a learning rate of 1.83×10^{-5}, dropout of 0.15, weight decay of 0.024, and 2 epochs. This configuration achieved our lowest validation loss and best F1 score.

DistilBERT Experiments:

- We reused the best hyperparameters from the BERT Optuna run with `distilbert-base-uncased`. While slightly worse in terms of F1 score and validation loss, DistilBERT was significantly faster and still competitive in performance.

3.1.1. Table of trials.

Trial	Model	Config Notes	Epochs	Val Loss / F1
1	BERT	lr=3*e*-5, batch=32	4	0.373 / 0.847
2	BERT	lr=3*e*-5, batch=32, wd=0.02	3	0.365 / 0.846
3	BERT (Optuna)	lr=1.83*e*-5, dropout=0.15, wd=0.024	2	**0.346 / 0.847**
4	DistilBERT (Optuna)	same as Trial 3	2	0.356 / 0.843

Table 1: Summary of main trials

3.2. Hyper-parameter tuning

Hyperparameters played a crucial role in balancing performance and overfitting. Our early brute-force trials showed signs of overfitting: the validation loss would stagnate or increase while training loss decreased.

To address this, we introduced weight decay and dropout regularization. Additionally, we limited the number of epochs and employed early stopping. The final model hyperparameters were tuned using the `Optuna` optimization framework, which performed a structured search over the space of learning rate, dropout, batch size, weight decay, and warmup ratio.

Key hyperparameters:

- **Learning Rate:** Best value was 1.83×10^{-5}.
- **Dropout:** Tuned to 0.1468 for both attention and hidden layers.
- **Weight Decay:** Optimal value was approximately 0.024.
- **Batch Size:** Best results achieved with 16.
- **Warmup Proportion:** 0.1799 to stabilize initial training.

These choices reduced overfitting while maintaining high validation performance.

3.3. Optimization techniques

For optimization, we used the AdamW optimizer, which combines Adam with decoupled weight decay regularization. This helps prevent overfitting while preserving adaptive learning rates. The optimizer was configured as follows:

- `lr=1.827e-5`, `eps=1e-8`, and `weight_decay=0.024`.

Gradient accumulation was employed with `GRADIENT_ACCUMULATION_STEPS=2` to effectively use larger batch sizes with limited GPU memory.

Additionally, early stopping with patience 2 was used to prevent unnecessary training epochs when validation performance plateaued. Warmup proportion of approximately 18% of the training steps was used to prevent instability in early training.

All hyperparameter optimization was automated using `Optuna`, which helped us converge to the best performing setup without manual trial-and-error.

3.4. Evaluation

In this section, we evaluate the model's performance using multiple metrics: **Accuracy, Precision, Recall, and F1 Score**. These metrics help assess the effectiveness of the classifier in distinguishing between positive and negative sentiments.

Note: While we only provide plots and detailed evaluation for the BERT model, the DistilBERT model exhibited almost identical learning curves and classification performance, with a slightly higher validation loss (0.3563 vs 0.3462) and an F1 score only 0.0046 lower. Due to time constraints and the minimal difference, we chose not to duplicate the plots for DistilBERT.

3.4.1. Evaluation Metrics. To define the key evaluation metrics, let's denote:

- **TP (True Positives)**: Correctly predicted positive instances.
- **TN (True Negatives)**: Correctly predicted negative instances.
- **FP (False Positives)**: Incorrectly predicted as positive.
- **FN (False Negatives)**: Incorrectly predicted as negative.

The evaluation metrics are defined as follows:

- **Accuracy**: Measures the proportion of correctly classified instances out of the total instances.
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$
A high accuracy indicates good performance, but it can be misleading for imbalanced datasets.

- **Precision**: Measures the accuracy of positive predictions.
$$Precision = \frac{TP}{TP + FP} \tag{2}$$
A higher precision means fewer false positives, which is useful in applications where incorrect positive classifications are costly (e.g., spam detection).

- **Recall**: Measures the ability to correctly identify all relevant instances.
$$Recall = \frac{TP}{TP + FN} \tag{3}$$
A high recall value ensures fewer false negatives, which is critical in scenarios like medical diagnosis.

- **F1 Score**: Provides a balance between precision and recall, especially useful in imbalanced datasets.
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

3.4.2. Model Performance Results. Table 2 summarizes the final model's evaluation metrics:

Metric	**Score**
Accuracy	0.8495
Precision	0.8605
Recall	0.8342
F1 Score	0.8472

Table 2: Final model performance metrics

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Negative (0)	0.84	0.86	0.85	21197
Positive (1)	0.86	0.83	0.85	21199
Macro Avg	0.85	0.85	0.85	42396
Weighted Avg	0.85	0.85	0.85	42396

Table 3: Classification report of the final model

3.4.3. Classification Report. The detailed classification report for each sentiment class (positive and negative) is presented in Table 3:

3.4.4. ROC Curve. The Receiver Operating Characteristic (ROC) curve illustrates the trade-off between the true positive rate (sensitivity) and the false positive rate. The area under the curve (AUC) represents the model's ability to differentiate between classes. It is visualized in Figure 5.

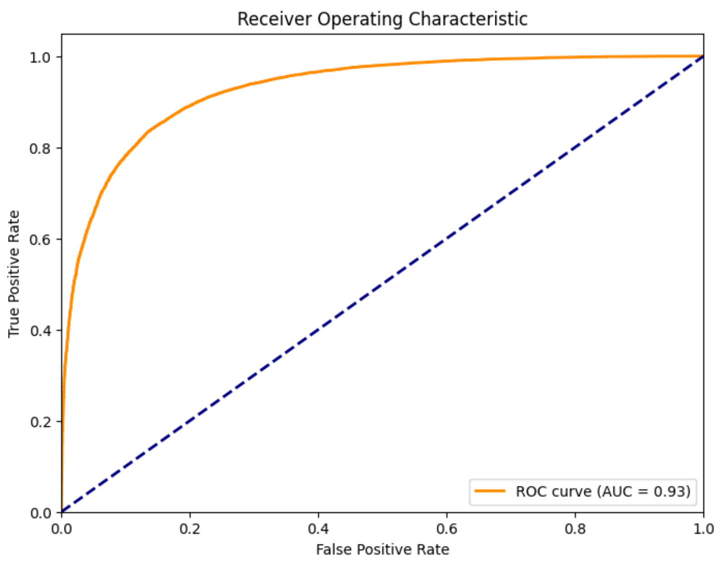

Figure 5: ROC Curve

3.4.5. Learning Curve. A learning curve helps in understanding the model's performance across different training set sizes. It allows us to diagnose underfitting or overfitting. It is visualized in Figure 6.

3.4.6. Confusion Matrix. The confusion matrix provides a summary of the prediction results. It highlights the number of correct and incorrect classifications. It is visualized

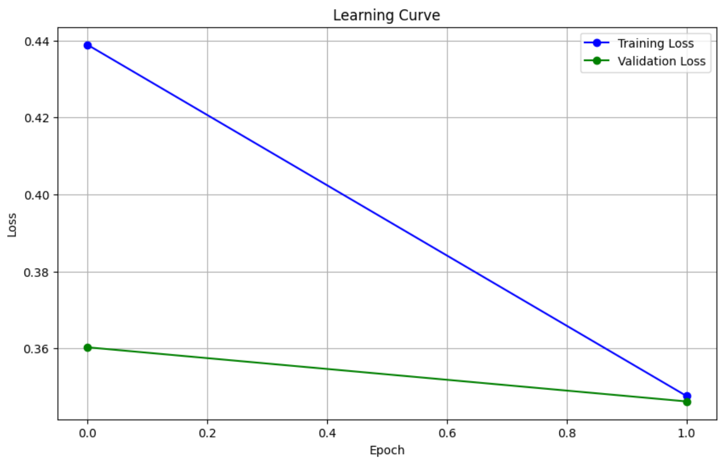

Figure 6: Learning Curve

in Figure 7.

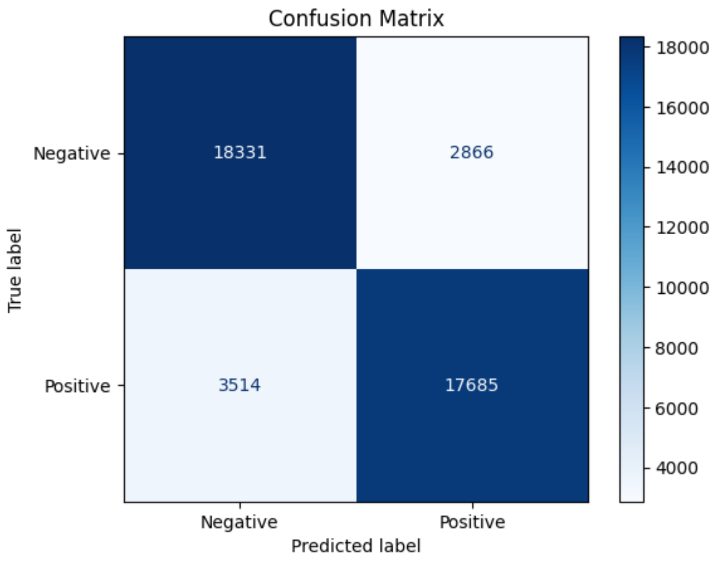

Figure 7: Confusion Matrix

Observations:

- The model achieved an **accuracy of 84.95%**, indicating strong performance on the sentiment classification task.
- The **precision (86.05%) and recall (83.42%)** demonstrate a good balance between minimizing false positives and false negatives.
- The **F1-score (84.72%)** confirms this balance, reinforcing the model's overall effectiveness.
- Given that the dataset has an equal distribution of positive and negative samples, accuracy is a meaningful and trustworthy evaluation metric in this context.

Overall, the evaluation results suggest that the model is well-optimized and performs consistently across different sentiment categories.

4. Results and Overall Analysis

4.1. Results Analysis

The final evaluation of our model was conducted on the validation dataset, which was distinct from the test dataset used during development. The model achieved an **accuracy of 84.95%** along with balanced precision and recall scores for both classes, as shown in Table 2. This demonstrates that the model can effectively distinguish between negative and positive sentiment with an F1 score of 0.8472, indicating strong overall performance.

The detailed classification report in Table 3 further confirms that both the negative and positive classes have comparable precision, recall, and F1-scores around 0.85, reflecting balanced performance and no major bias towards either class.

4.1.1. Training and Validation Trends. The training process consisted of two epochs, each taking approximately 28 minutes on a GPU. Table 4 summarizes the training and validation metrics per epoch, showing steady improvement from the first to the second epoch in terms of loss and all performance metrics.

Epoch	Train Loss	Val Loss	Accuracy	Precision	Recall
1	0.4389	0.3603	0.8414	0.8368	0.8483
2	0.3476	0.3462	0.8495	0.8605	0.8342

Table 4: Training and validation statistics per epoch

4.1.2. Best Trial. The best trial was obtained using Optuna with the following configuration:

- Learning Rate: 1.83×10^{-5}
- Batch Size: 16
- Weight Decay: 0.024
- Dropout Rate: 0.15
- Number of Epochs: 2
- Warmup Proportion: 0.18
- Model: bert-base-uncased

This configuration yielded the lowest validation loss and highest overall performance, making it the optimal setup for this project.

4.2. Comparison with the first project

In this project, we developed a sentiment classifier using a transformer-based architecture, specifically BERT, fine-tuned on the English-language Twitter dataset. Unlike the first project, which relied on a simpler TF-IDF + logistic regression pipeline, our approach leverages contextual language understanding via pretrained transformer embeddings.

4.2.1. Experiments. In the first project, we used logistic regression with TF-IDF vectorization, achieving 80.47% accuracy. In this project, we fine-tuned a BERT model using PyTorch, reaching a significantly higher validation accuracy of 84.95%. Key experimental differences include:

- **Feature Representation:**
  - Project 1: TF-IDF (bag-of-words)
  - Project 3: BERT embeddings (contextual, sentence-level)
- **Model Architecture:**
  - Project 1: Logistic regression (linear)
  - Project 3: Pretrained BERT encoder + classification head (transformer-based)
- **Hyperparameter Tuning:**
  - Project 1: Manual grid search
  - Project 3: Optuna hyperparameter tuning (5 trials)

Metric	**Project 1 (TF-IDF + LR)**	**Project 3 (BERT)**
Accuracy	80.47%	**84.95%**
Precision	80.00%	**86.05%**
Recall	81.00%	**83.42%**
F1 Score	80.50%	**84.72%**
Training Time	5 minutes	1 hour

4.2.2. Analysis. Here are the main strengths and limitations of our BERT-based approach compared to the baseline TF-IDF model:

- **Strengths:**
  - BERT captures contextual and semantic nuances in sentences, not just individual keywords.
  - Fine-tuning leverages pretrained knowledge from large corpora, improving generalization.
  - Performance improved across all key metrics (accuracy, precision, F1).
  - Robust to token variation (e.g., casing, misspellings) due to WordPiece tokenization.
- **Limitations:**
  - Requires significantly more computational resources (GPU memory and time).
  - More complex to implement and fine-tune compared to traditional models.
  - Less interpretable than linear models.

Despite its increased complexity, our BERT-based approach outperformed the simpler TF-IDF + logistic regression pipeline in all metrics. The accuracy gain of over 4% shows that deep contextual embeddings provide a major advantage for sentiment classification, particularly when classifying nuanced or ambiguous tweets.

The success of our approach can be attributed to:

- BERT's ability to understand word meaning based on surrounding context, unlike TF-IDF which treats words independently.
- Fine-tuning on our dataset, which allows the model to adapt pretrained weights to domain-specific language.
- Use of Optuna for efficient and systematic hyperparameter search.

4.2.3. Plots.

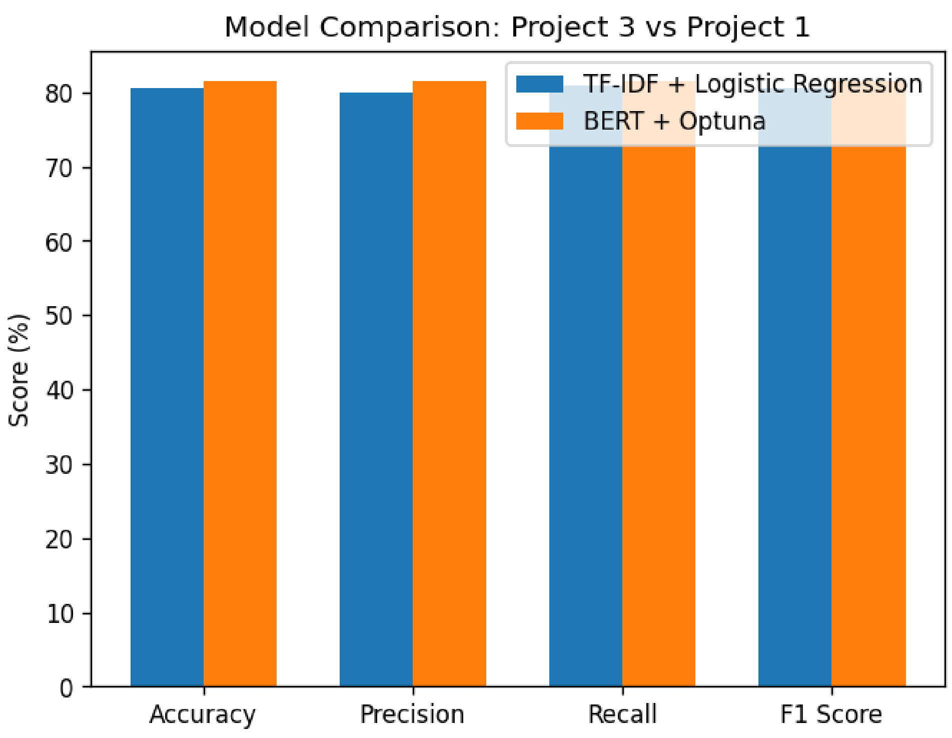

Figure 8: Performance comparison between Project 1 (blue) and Project 3 (orange) across metrics

4.2.4. Conclusion. Our project demonstrates that transformer-based models like BERT can significantly outperform traditional machine learning models on sentiment classification tasks, even with moderate dataset sizes (~200k samples). While TF-IDF is fast and simple, BERT's contextual understanding offers a distinct advantage in modern NLP pipelines.

4.3. Comparison with the second project

In this project, we implemented a sentiment classifier based on the BERT transformer architecture, fine-tuned on an English Twitter dataset. Compared to Project 2, which used a 2-layer neural network with Word2Vec embeddings, our model achieved notably better performance across all evaluation metrics.

4.3.1. Experiments. Project 2 used Word2Vec embeddings averaged per sentence and fed into a shallow neural network. While this captured some semantic information, it ignored word order and context. Our project instead fine-tunes a BERT encoder that understands contextual relationships and token dependencies.

Key differences include:

- **Feature Representation:**
  - Project 2: Averaged Word2Vec embeddings (300-dimensional)
  - Project 3: Contextual BERT embeddings (768-dimensional, pretrained on BookCorpus and Wikipedia)

- **Model Architecture:**
  - Project 2: 2-layer feedforward neural network with ReLU and dropout
  - Project 3: Pretrained BERT encoder + linear classification head
- **Hyperparameter Tuning:**
  - Project 2: Optuna tuning (30 trials)
  - Project 3: Optuna tuning (50 trials)

Metric	**Project 2 (Word2Vec + NN)**	**Project 3 (BERT)**
Accuracy	75.84%	**84.95%**
Precision	73.60%	**86.05%**
Recall	80.70%	**83.42%**
F1 Score	77.00%	**84.72%**
Training Time	4 hours	1 hour

*4.3.2. **Analysis.*** Our BERT-based classifier clearly outperforms the Word2Vec-based neural network from Project 2 across all dimensions. The improvement is particularly pronounced in precision and overall accuracy, which suggests that BERT is better at correctly identifying sentiment-bearing patterns in noisy Twitter text.

- **Strengths of our approach:**
  - BERT captures contextual meaning and token interactions, unlike static Word2Vec vectors.
  - Fine-tuning leverages large-scale pretraining and adapts it to task-specific data.
  - Better generalization and robustness across sentiment boundaries (neutral/ambiguous cases).
  - Faster convergence using GPU and mixed precision.
- **Limitations of Project 2:**
  - Mean-pooling over Word2Vec embeddings ignores syntax and word order.
  - Case-sensitive embedding lookup caused frequent out-of-vocabulary issues.
  - Neural network required extensive tuning and still underperformed.

The 9% increase in accuracy and nearly 8% gain in F1-score demonstrate the advantage of contextualized deep learning models over shallow architectures with static embeddings.

*4.3.3. **Plots.***

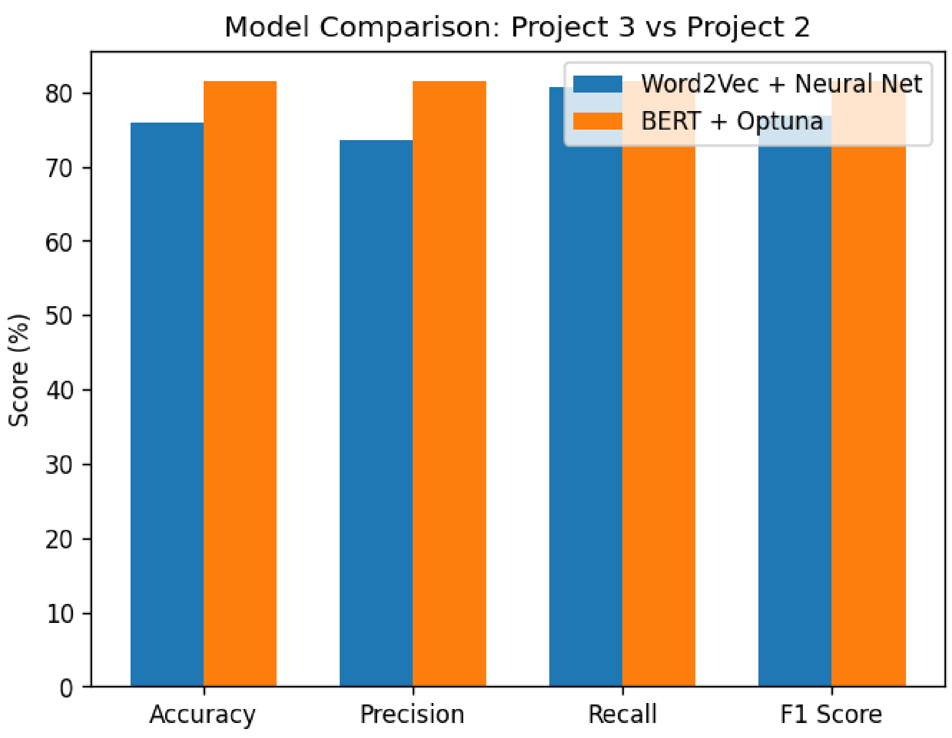

Figure 9: Performance comparison between Project 2 (blue) and Project 3 (orange) across metrics

4.3.4. Conclusion. This comparison confirms that pretrained transformer models such as BERT provide a significant upgrade in both performance and modeling capability over traditional static embeddings and shallow networks. While Project 2 introduced deeper architectures compared to logistic regression, it lacked the ability to capture sentence-level meaning and word interactions.

5. Code Availability

The full code for this project is available in the following Kaggle notebooks:

- BERT Notebook
- DistilBERT Notebook

Please note that these notebooks are shared privately with the teaching assistant, Despina-Athanasia Pantazi (`dpantazi@di.uoa.gr`), and are not publicly accessible.

6. Bibliography

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[1]