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 classifier using Logistic Regression with TF-IDF method features on a given Twitter training dataset. The dataset consists of tweets labeled as either negative(0) or positive(1). To tackle this task, I will first perform Exploratory Data Analysis (EDA) to understand the dataset and its characteristics. Then, I will preprocess the text data by applying a necessary function which cleans the data. Next, I will extract features using the TF-IDF method and train a Logistic Regression model. Model performance will be evaluated using accuracy, precision, recall, and F1-score. Finally, I will fine-tune the model and analyze its results through visualizations and plots.

2. Data processing and analysis

2.1. Pre-processing

Before using TF-IDF method and training a Logistic Regression model, pre-processing the text data is an important step to improve models's accuracy. The following methods I used for data cleaning and pre-processing are:

1. Lowercasing: Converting all text to lowercase ensures that words like "The" and "the" are treated the same, reducing redundancy in the vocabulary.
2. Tokenization: Splitting the text into words or phrases based on Twitter-specific text
3. Contractions: Converting contractions like "isn't" to "is not" for better clarity in word representation.
4. Removing mentions and links: Eliminating Twitter mentions (@user) 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: Standardizing elongated words (e.g., "plzzzz" to "plz") and extra spaces to maintain uniformity.
7. Slang normalization: Replacing common informal words with their standard equivalents to improve text understanding.

These preprocessing steps help refine the dataset, reducing variability and noise, ultimately leading to a more effective sentiment classification model.

2.2. Analysis

To gain a comprehensive understanding of the dataset, I firstly counted all unique words, revealing 234,492 distinct features (see Figure 1). Additionally, I analyzed the class distribution, finding 148,388 total tweets, with an even split between positive

(50%) and negative (50%) tweets (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. Next, I examined the most frequent unigrams, bigrams, and trigrams to identify common linguistic patterns. A key observation was the high prevalence of URLs, which introduce noise into the dataset. To enhance feature quality, URLs will be removed during preprocessing to ensure the model focuses on meaningful text rather than web links (see Figure 3). Finally, an analysis of top unigrams, bigrams, and trigrams on positive and negative tweets seperatedly was conducted. Positive tweets often include phrases such as "thank you", "good day", and "happy birthday", reflecting gratitude, greetings, and celebrations. In contrast, negative tweets frequently contain "feel like", "don't like", and "not happy", indicating dissatisfaction or complaints.
These insights from the EDA phase helped refine the preprocessing steps and provided an initial understanding of how sentiment is expressed in the dataset. EDA was conducted 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. For these reasons, no EDA was performed on the validation or test dataset.

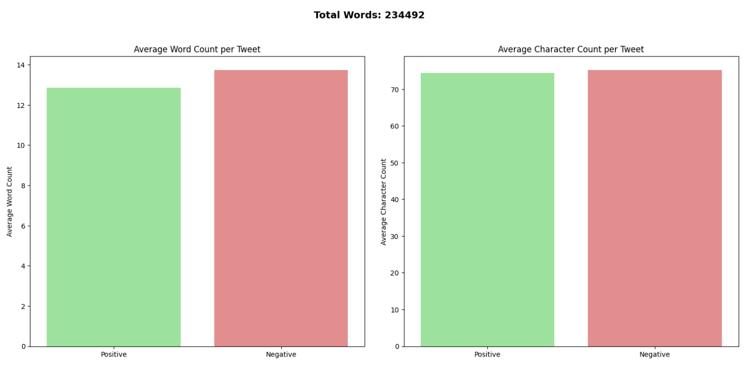


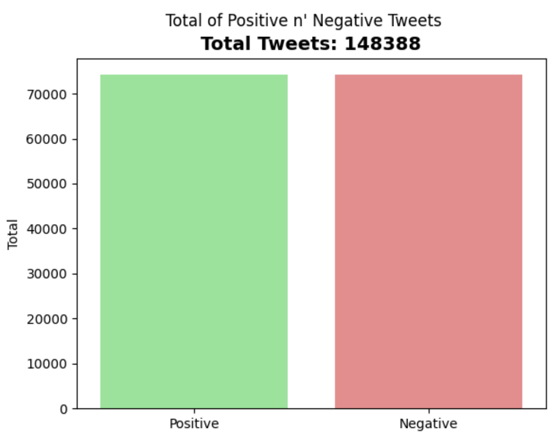
Figure 1: Average Word and Character Count per Tweet

2.3. Data partitioning for train, test and validation

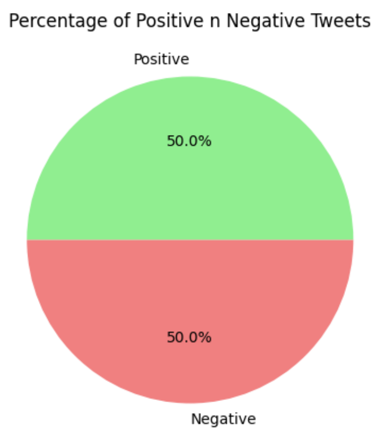
To effectively train and evaluate the sentiment classification model, the dataset was partitioned into three subsets: training set, validation set, and test set, as it was shown in the lectures. The dataset was split as follows:

1. Training Set (X_train, y_train): This set consists of 148,388 tweets, as the EDA showed, and is used to train the model by learning patterns from labeled data.

2. Validation Set (X_val, y_val): This set is used to fine-tune the model by selecting optimal hyperparameters and avoiding overfitting. It helps assess performance before making predictions on the test set.



(a)



(b)

Figure 2: Comparison of Positive and Negative Tweets

3. Test Set (X_test): This set contains unseen tweets and is used to evaluate the performance of the final model.

2.4. Vectorization

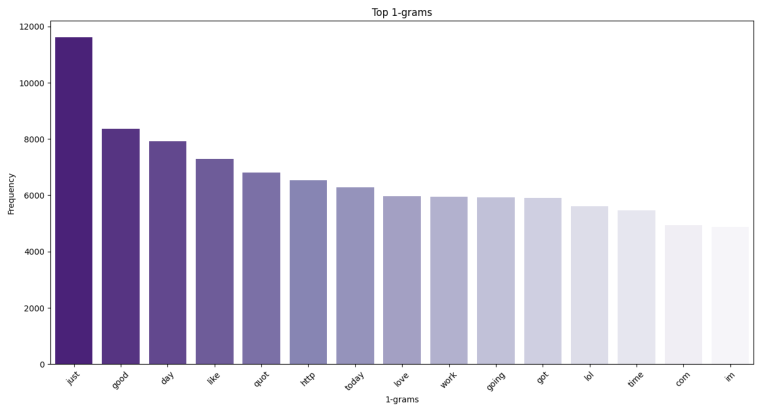
To convert textual data into numerical representations suitable for machine learning, TF-IDF (Term Frequency-Inverse Document Frequency) was used as the vectorization technique. This method, implemented using Scikit-learn's TfidfVectorizer from sklearn.feature_extraction.text, transforms text into numerical feature vectors by evaluating the importance of words in the data set.

So, how TF-IDF Works:
Term Frequency (TF) measures how often a word appears in a document and Inverse Document Frequency (IDF) assigns a weight to words based on their rarity across all documents, reducing the influence of common words. As a result, TF-IDF Score is computed as

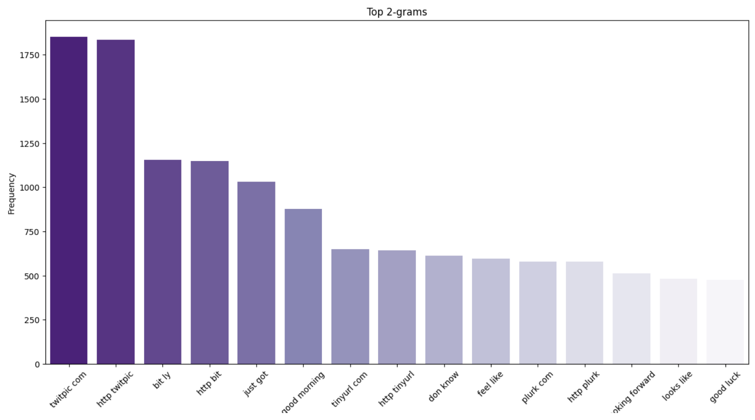
$$TF - IDF = TF \cdot IDF. \tag{1}$$

Vectorization Configuration:
The TfidfVectorizer function was configured with the following parameters to optimize performance:

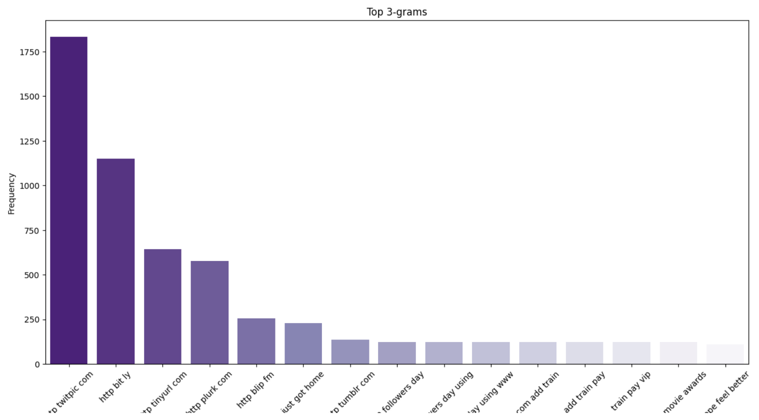
1. ngram_range=(1,3): It captures words, bigrams, and trigrams for better contextual understanding.
2. min_df=3 : It ignores words that appear in fewer than three documents to reduce noise. This parameter was set through tests.
3. max_features=50000: The vocabulary size is limited to 50,000 words, approximately 20% of the total 250,000 words, to reduce dimensionality and mitigate overfitting, ensuring a more efficient and generalized model.



(a)

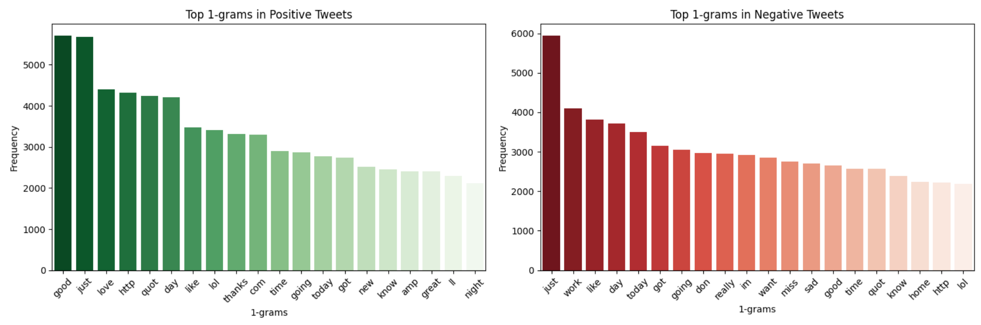


(b)

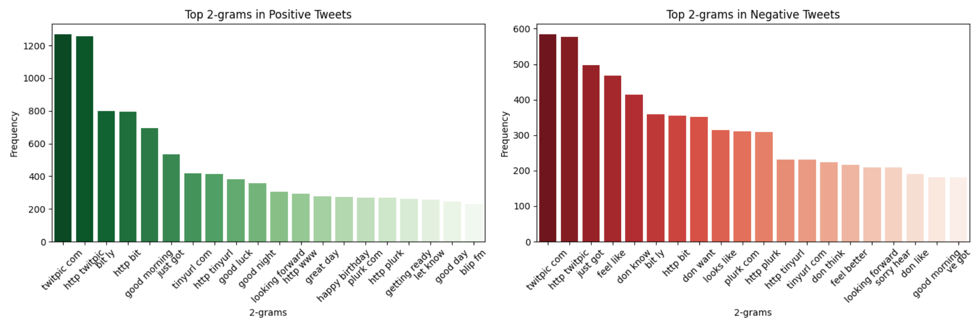


(c)

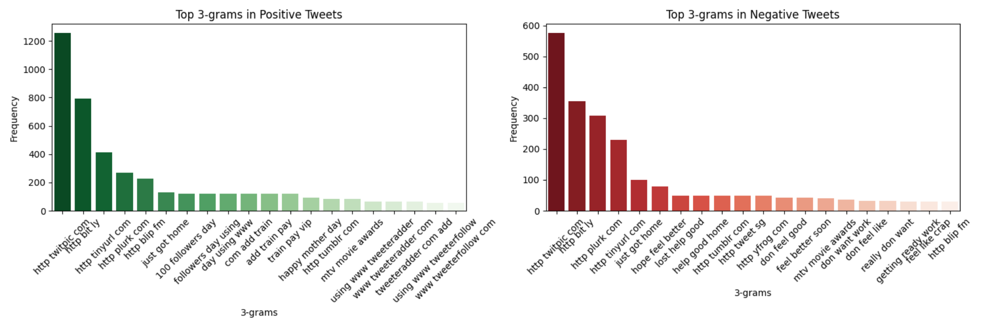
Figure 3: Top 15 most used words, bi-grams, tri-grams



(a)



(b)



(c)

Figure 4: Top 10 most used words, bi-grams, tri-grams per Sentiment

4. stop_words=["and", "is", "are", "i"]: It removes selected stopwords that negatively impacted accuracy.

3. Algorithms and Experiments

3.1. Experiments

1. For the first attempt, I used a basic Logistic Regression model with TF-IDF vectorization, without applying any preprocessing or hyperparameter tuning. This resulted in a baseline accuracy of 0.79114. The learning curve (shown below) indicates overfitting, the training accuracy is significantly higher than the validation accuracy. Specifically, the training score starts high (0.88) but decreases slightly as more data is added. The validation score remains consistently lower (0̃.74–0.78), showing that the model struggles to generalize. As a result, the gap between training and validation performance suggests that the model is memorizing patterns in the training set rather than learning generalizable features.

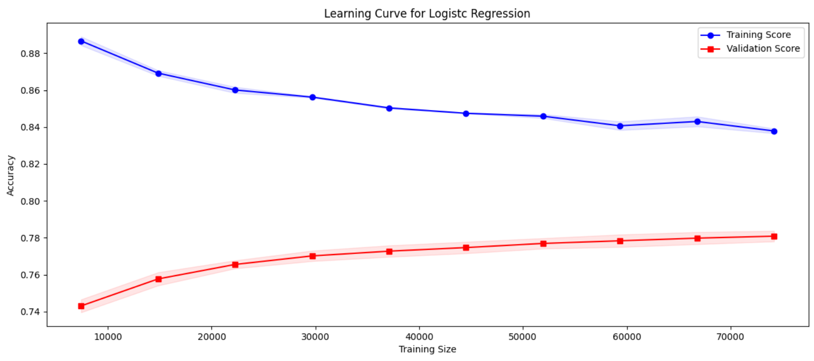


Figure 5: Plot for baseline Logistic Regression used with TF-IDF method

2. For the first optimization attempt, I introduced slang normalization and lowercasing to the text pre-processing pipeline. However, instead of improving performance, these modifications led to a slight decrease in accuracy, dropping to 0.79040.

3. To improve the model's accuracy, I utilized the ngram_range parameter of the TfidfVectorizer function, setting it to (1,3). This adjustment led to an accuracy increase of 0.79595, as the model was better able to recognize and treat multi-word phrases, such as "New York," as distinct entities. Another parameter I adjusted in the TfidfVectorizer function was min_df, which I set to 3. This change resulted in an accuracy increase of 0.80243, as it prevented the model from giving undue importance to words that appeared only once and twice, thereby improving its focus on more meaningful terms.

4. Initially, I set stop_words='english' in the TfidfVectorizer function, assuming that removing common words would improve the model's accuracy by eliminating

less informative terms. However, this led to a decrease in accuracy by 0.77128, likely because some removed words carried contextual importance for classification. To improve the results, I tested custom stopword lists individually, which resulted in an accuracy increase to 0.80267. Finally, I set the max_features parameter to 50,000, which boosted the accuracy to 0.80321 by focusing the model on the most important words and avoiding overfitting.

5. To enhance the model's accuracy, additional preprocessing steps were applied to the text in the training dataset, as detailed in Section 2.1. These preprocessing techniques included tokenization, contraction expansion, removal of Twitter mentions and URLs, and space normalization. As a result, the model's accuracy improved to 0.80519. However, one preprocessing step—removing special characters and numbers—unexpectedly led to a decrease in accuracy. While this technique is typically beneficial for reducing noise in text data, it may have inadvertently removed valuable information. Special characters, such as punctuation, could carry sentiment-related context (e.g., "great!!!" vs. "great"), while numbers might be relevant in certain tweets (e.g., "5-star service"). By eliminating these elements, the model potentially lost meaningful features that contributed to distinguishing between positive and negative sentiments, ultimately impacting its performance.

6. Finally, Grid Search with 5-fold cross-validation was applied to identify the optimal hyperparameters for Logistic Regression. Specifically, the search focused on regularization strength (C), solver type, and the number of iterations. This optimization process improved the model's accuracy to 0.80536. The reason behind the selection of these parameters is further discussed in Section 3.2.

Trial	Improvement	Score
1	Baseline of Logistic Regression with TF-IDF method	0.79114
2	Lowercasing & Slang Normalization	0.79040
3	Set ngram_range to (1,3) & min_df to 3	0.80243
4	Set Stopwords & max_features to 50,000	0.80321
5	Tokenization, Contractions, Removal of mentions, URLs & multiple spaces	0.80519
6	Grid Search with 5-fold cross validation	0.80536

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning

The final result of my model was accuracy on 0.80536 for the validation dataset.
For the model configuration, I utilized the TF-IDF method for feature extraction and Logistic Regression as the classifier. Before applying TF-IDF, I implemented a preprocessing function on the training dataset to clean the text and enhance the quality of input data. This step ensures that the model focuses on meaningful patterns rather than noise. The specific TF-IDF parameters used are detailed in Section 3.1.
To optimize model performance, Grid Search with cross-validation was used to find

the best hyperparameters for regularization in Logistic Regression. The optimal configuration found was:

- C = 2: The regularization strength controls the balance between simplicity and complexity in the model. A value of 2 helps prevent underfitting (being too simple) while avoiding overfitting (being too complex), ensuring better generalization to new data.
- Solver = 'liblinear': This algorithm is well-suited for large-scale datasets and efficient for binary classification.
- Max_iter = 100: This defines the maximum number of iterations for the solver to converge. Based on experimentation, 100 iterations provided a good balance between training time and achieving optimal convergence.

These hyperparameter choices helped improve generalization, reducing overfitting while maintaining high predictive accuracy on unseen data.

Overfitting and Underfitting,
Initially, the model exhibited overfitting, with significantly higher training accuracy than validation accuracy, suggesting that the model was memorizing patterns rather than generalizing well to unseen data. After hyperparameter tuning and preprocessing enhancements, the gap between the two scores gradually decreased, indicating improved generalization. The learning curve confirms this, as training and validation accuracy converge with more training data. In the provided plot(see Figure 6) the training accuracy gradually decreases but remains stable and the validation accuracy steadily increases and gets closer to the training accuracy. Lastly, no signs of underfitting are observed, as the final accuracy score is sufficiently high in order to suggest that the model effectively captures meaningful patterns.

3.3. Optimization techniques

To enhance model performance, several optimization strategies were applied using Scikit-learn's optimization frameworks:

1. Data Preprocessing:
   - Lowercasing: to standardize text.
   - Tokenization using a Twitter-specific tokenizer by NLTK (Natural Language Toolkit) library.
   - Contraction expansion: (e.g., "isn't" $\rightarrow$ "is not") by Contractions library .
   - Removal of noise such as mentions (@username), URLs, punctuation, multiple spaces by re module.
   - Character normalization (e.g., "plzzzz" $\rightarrow$ "plz").
   - Slang normalization: improve text quality (e.g., "cuz" $\rightarrow$ "because").
2. TF-IDF Feature Engineering
   Used TfidfVectorizer from sklearn.feature_extraction.text for transforming text into numerical representations:

- Stopwords selection: Initially tested a broader list, but a smaller set led to better accuracy.
- N-grams: Specifically (1 to 3) to capture context better.
- Minimum document frequency: Set to (min_df = 3) to filter rare words.
- Max features: Tuned to (50,000) to balance information retention and computational efficiency.

3. Hyperparameter Tuning (Grid Search):
   A Grid Search with 5-fold cross-validation was conducted to optimize Logistic Regression parameters, used from Scikit-learn's GridSearchCV from sklearn.model_selection:

   - C (Regularization Strength): Tested values [0.1, 0.5, 1, 2], with C = 2 yielding the best balance between bias and variance.
   - Solver: Compared 'liblinear' and 'lbfgs', with 'liblinear' performing better.
   - Max Iterations: Tested [100, 250, 500], with 100 being sufficient for convergence.

These optimizations significantly enhanced model generalization, reducing overfitting while maintaining high accuracy.

3.4. Evaluation

To assess the effectiveness of the model, 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.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

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

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

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

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

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

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

	Precision	Recall	F1-score	Support
Class 0	0.81	0.80	0.80	21197
Class 1	0.80	0.81	0.81	21199
Accuracy			0.81	42396
Macro avg	0.81	0.80	0.81	42396
Weighted avg	0.81	0.81	0.81	42396

Table 2: Evaluation Metrics

4. Results and Overall Analysis

4.1. Results Analysis

The final model achieved an accuracy of 0.80536 on the validation set on Kaggle and 0.80371 on the test test, demonstrating a solid performance for a Logistic Regression classifier with TF-IDF features. Given the simplicity of the model compared to more advanced deep learning approaches, this accuracy is reasonable and aligns with expectations.

Evaluation Metrics Overview

- Accuracy: 0.80536
- Precision: 0.80074
- Recall: 0.81306
- F1-Score: 0.80685

The balanced precision and recall indicate that the classifier does not overly favor one class over another, leading to an F1-score of approximately 0.81. This suggests that the model generalizes well and maintains a good trade-off between false positives and false negatives.

Learning Curves
The provided learning curve visually represents the model's training and validation performance as the training data size increases. The training accuracy (blue curve) starts high but gradually decreases as more data is introduced. The validation accuracy (red curve), on the other hand, increases steadily, demonstrating that the model is learning meaningful patterns and benefiting from more training data. The convergence between the two curves at higher training sizes indicates that overfitting has been reduced, meaning the model generalizes better to unseen data. However, the noticeable gap between training and validation accuracy suggests that the model still has room for improvement.
The experiments for this model have been thoroughly analyzed in Section 3.1, and given the findings, but there are several additional experiments and improvements that could be explored to further enhance the model's performance such as implementing the more advanced models.

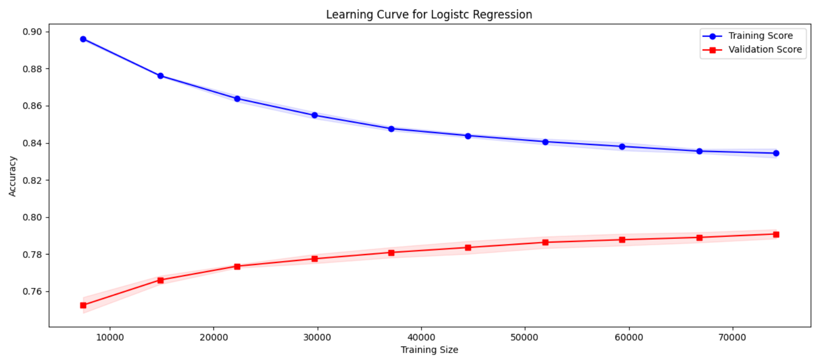


Figure 6: Plot for the best trial

4.1.1. Best Trial.

- Model Used
  - Logistic Regression Classifier with TF-IDF feautures
- Evaluation Metrics
  - Accuracy: 0.80536
  - Precision: 0.80074
  - Recall: 0.81306
  - F1-Score: 0.80685
- Pre-processing Steps
  - Data Cleaning and Text Normalization
  - Tf-IDF Vectorization
- Best Hyper-parameters
  - C = 2
  - solver = liblinear
  - max iterations = 100

5. Bibliography

References

[1] Manolis Koubarakis. Introductory concepts of machine learning. 2025.

[2] Manolis Koubarakis. Regression. 2025.

[3] Scikit-learn.org. Gridsearchcv.

[4] Scikit-learn.org. Logistic regression.

[5] Scikit-learn.org. Tfidfvectorizer.

[1] [2] [4] [5] [3]