University of Athens
Department of Informatics and Telecommunications

Deep Learning for NLP

Student name: *Nektarios Tsimpourakis Pavlakos*
sdi: sdi2200196

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

Contents

1 Abstract **2**

2 Data processing and analysis **2**
2.1 Pre-processing 2
2.2 Analysis 2
2.3 Data partitioning for train, test and validation 5
2.4 Vectorization 5

3 Algorithms and Experiments **6**
3.1 Experiments 6
3.1.1 Table of trials 6
3.2 Hyper-parameter tuning 7
3.3 Optimization techniques 8
3.4 Evaluation 9
3.4.1 ROC curve 9
3.4.2 Learning Curve 10
3.4.3 Confusion matrix 10

4 Results and Overall Analysis **11**
4.1 Results Analysis 11
4.1.1 Best trial 11

5 Bibliography **12**

1. Abstract

This assignment requires the development of a sentiment classifier using only Logistic Regression and only TF-IDF in Python on a given English-language Twitter dataset. My implementation will go through these main steps: Exploratory Data Analysis (EDA), Text Preprocessing, Feature Extraction, Model Development and Evaluation

2. Data processing and analysis

2.1. Pre-processing

The data cleaning techniques that were found to be the most beneficial and thus used in the final implementation are as follows:

- Conversion to lowercase.
- Anonymization through the removal of tags and links.
- Targeted correction of certain grammatical errors.
- Removal of punctuation.
- Removal of non-ASCII characters.
- Removal of repeated characters (3 or more times $\rightarrow$ 1 occurrence).
- Replacement of multiple spaces with a single space.

2.2. Analysis

The dataset consists of X training examples, Y validation examples and Z test examples. The class distribution shows that positive tweets account for $\approx$ 51% of the data, while negative tweets make up $\approx$ 49%, indicating a balanced dataset (see Figure 1).

To better understand the dataset, we visualized the distribution of text lengths. The histogram showed that most tweets range from 50 to 120 characters (see Figure 2). Additionally, analyzing common n-grams before preprocessing revealed that 'good', 'just', 'just' and 'love' are among the most frequent words in positive tweets (see Figure 3), while 'just', 'work', 'miss' and 'day' are dominant in negative tweets (see Figure 4). It's worth mentioning how dominant the word 'just' is for both types of tweets.

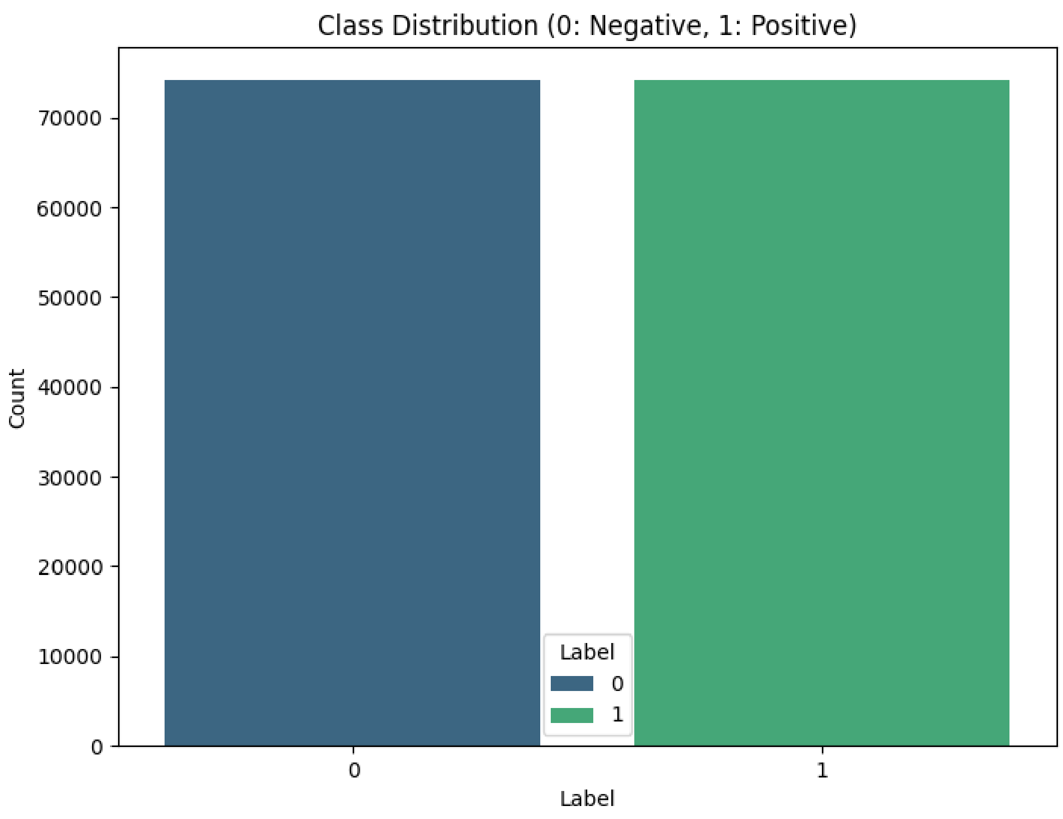

Figure 1: Class Distribution based on sentiment

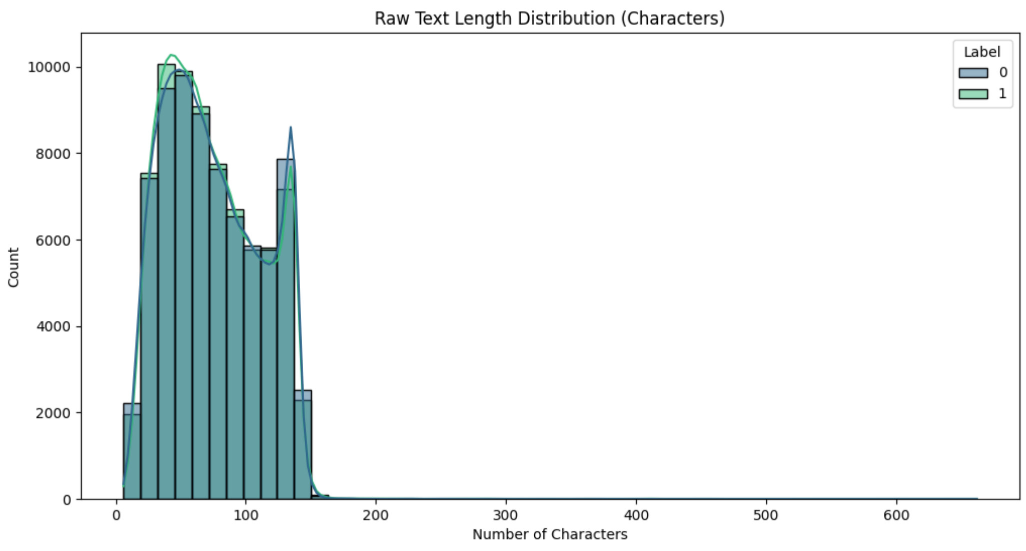

Figure 2: Raw Text Length Distribution

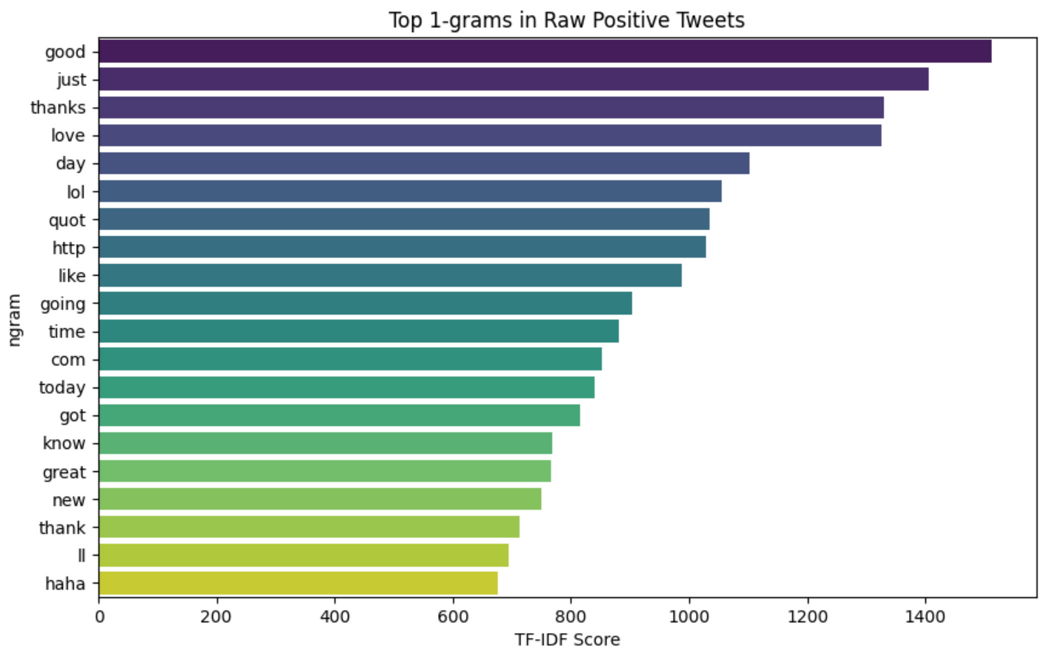

Figure 3: Top 1-grams in Raw Positive Tweets

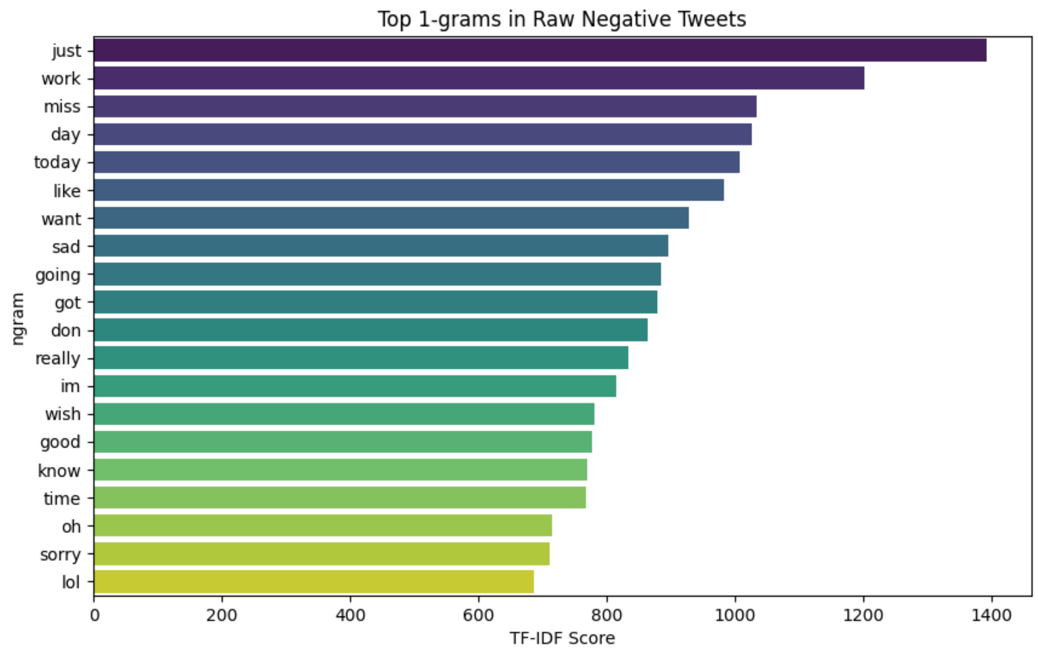

Figure 4: Top 1-grams in Raw Negative Tweets

2.3. Data partitioning for train, test and validation

The dataset was partitioned into three subsets: training, validation, and test. The goal of this partitioning is to train the model, fine-tune it, and finally evaluate its performance on unseen data to ensure its generalizability.

1. Training Set: The model was trained using the data in the train_df file, which contains both the text and the corresponding labels. The training data is used to fit the model, allowing it to learn the patterns and relationships between the text features and the sentiment labels.

2. Validation Set: The val_df file, which also contains both text and labels, was used for model validation. The validation set helps in evaluating the model's performance during training. It acts as an intermediate stage between training and testing.

3. Test Set: The final evaluation of the model was performed on the test_df file. This set contains only the text column since the labels are not available. The predicted labels are then compared to the actual test labels found when uploading the final submission.

2.4. Vectorization

The evaluation of the text data is succeeded through TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. TF-IDF is a statistical measure used in natural language processing and information retrieval to evaluate the importance of a word in a document relative to a collection of documents.

- TF: Measures how often a word appears in a document. A higher frequency suggests greater importance.

- IDF: Reduces the weight of common words across multiple documents while increasing the weight of rare words.

For this analysis, I used the TfidfVectorizer from scikit-learn, with the following parameters:

- ngram_range=(1, 3): This considers unigrams (single words), bigrams (pairs of words), and trigrams (triplets of words) as features, allowing the model to capture context and relationships between adjacent words.

- stop_words=my_stop_words: A custom list of common English stop words (e.g., "the", "and", "is") was provided to remove uninformative words that do not contribute much to the model's predictive power.

- max_features=75000: The number of features was limited to the top 75000 most informative words and n-grams.

- min_df=2: Words that appear in fewer than two documents were excluded from the vocabulary to reduce the impact of overly rare words.

3. Algorithms and Experiments

3.1. Experiments

To tackle this problem, I started with a baseline model, inspired by the workshop conducted by Yorgos Pantis, which achieved an initial accuracy of 77%. From there, I incrementally refined my approach by experimenting with different techniques to improve performance.

Text Preprocessing
One of the first areas I focused on was the `preprocess_text` function. By continuously inspecting the processed text, I identified and addressed errors or inefficiencies in the tokenization and cleaning steps.
While some adjustments improved accuracy and led to the final state of the function (see subsection 2.1) , certain attempts at vocabulary correction and contraction handling (for English text) failed, either having a negligible effect or even reducing model performance.

Vectorization Adjustments
After refining text preprocessing, I experimented with different vectorization parameters. This involved:

- Adjusting the n-gram range to capture context more effectively.
- Experimenting with removing a custom list of common English stop words that were of no use to the model.
- Adjusting the max_features so that overfitting could be suppressed.
- Tweaking the TF-IDF weighting to balance common and rare terms.

This process led to gradual improvements and led to the final vectorization parameters (see subsection 2.4)

Hyperparameter Tuning and Cross-Validation
To optimize the model, I conducted hyperparameter tuning using techniques such as cross-validation and GridSearchCV. However, GridSearchCV did not appear to be more efficient when compared to manual tuning. (see subsection 3.2)

3.1.1. Table of trials. The table below provides insights into the optimization process of my model, showcasing the experiments conducted through my successful Kaggle submissions.

Trial	Description	Score
1	Minimal preprocessing, suboptimal vectorization, no hyperparameters	0.80041
2	Improved preprocessing	0.80093
3	Improved preprocessing, adjustment of stop word list	0.80135
4	Improved preprocessing, Introduced hyperparameters	0.80319
5	Adjustments with ideal parameters	0.80319
6	Fixed logical error in preprocessing	0.80414
7	Adjustments to max_features	0.80470
8	Decrease of max_features to reduce overfitting, final model	0.80366

Table 1: Trials

3.2. Hyper-parameter tuning

After performing hyperparameter tuning, the best hyperparameters for the Logistic Regression model were found to be:

- C (Regularization Strength): 2
- Solver: lbfgs
- Penalty: l2

How the Model was Configured:

- **Regularization Strength (C)**:
  C=2 was chosen as the best value. A smaller C (e.g., C=0.01) would apply stronger regularization, potentially underfitting the data, while a larger C (e.g., C=100) would apply weaker regularization, potentially overfitting the data. C=2 strikes a balance between underfitting and overfitting.
- **Solver (lbfgs)**:
  The lbfgs solver was chosen because it is efficient for medium-sized datasets and supports the l2 penalty. It is also less prone to numerical instability compared to other solvers like sag or liblinear.
- **Penalty (l2)**:
  The l2 penalty (Ridge regularization) was chosen because it discourages large coefficients without forcing them to zero, which helps prevent overfitting while maintaining model flexibility.

Underfitting vs. Overfitting

- **Underfitting**:
  If the model is too simple (e.g., using a very small C value like C=0.01), it may underfit the data. This would result in poor performance on both the training and validation sets, as the model fails to capture the underlying patterns in the data. In this case, the model did not underfit, as evidenced by the relatively high validation accuracy.

- **Overfitting**:
  If the model is too complex (e.g., using a very large value for C like C=100 or for max_features like max_features=200000), it may overfit the data. This would result in high training accuracy but poor validation accuracy, as the model memorizes the training data instead of generalizing to unseen data. In this case, the model's tendency to overfit was handled accordingly, as evidenced by the similar performance on the training and validation sets (Performance Gap (Train - Val): 0.0606).

3.3. Optimization techniques

Summary of Optimization Techniques:

1. Hyperparameter Tuning: Grid search with cross-validation to find the best hyperparameters.
2. Regularization: L2 regularization to prevent overfitting.
3. Feature Engineering: TF-IDF vectorization with custom stop words and n-grams.
4. Text Preprocessing: Cleaning and standardizing the text data.
5. Learning Rate and Iterations: Ensuring the model converges.
6. Evaluation Metrics: Using accuracy, precision, recall, F1-score, ROC curve, and AUC to evaluate performance.
7. Learning Curve Analysis: Ensuring the model is not overfitting or underfitting.
8. Confusion Matrix: Visualizing the model's predictions.

Scikit-Learn: The primary optimization framework used for model training, hyperparameter tuning, and evaluation. Key modules used include:

1. LogisticRegression: For training the model.
2. TfidfVectorizer: For feature extraction.
3. GridSearchCV: For hyperparameter tuning.
4. cross_val_score: For cross-validation.
5. accuracy_score, precision_score, recall_score, f1_score: For evaluation metrics.
6. roc_curve, auc: For ROC curve analysis.
7. learning_curve: For generating the learning curve.
8. confusion_matrix: For visualizing the model's predictions.

3.4. Evaluation

To assess the performance of the trained logistic regression model, I used several classification metrics, including accuracy, precision, recall, and F1-score. These metrics provide insight into the model's effectiveness in predicting sentiment labels.

Evaluation Metrics

- Accuracy: Measures the proportion of correctly classified instances out of the total instances.
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
The validation accuracy achieved is 0.804887, meaning the model correctly predicts around 80.49% of the samples.
- Precision: Measures the accuracy of positive predictions
$$Precision = \frac{TP}{TP + FP}$$
The precision is 0.801793, indicating that when the model predicts a positive sentiment, it is correct 80.18% of the time.
- Recall: Measures the ability to capture all relevant instances
$$Recall = \frac{TP}{TP + FN}$$
The recall achieved is 0.810038, meaning the model correctly identifies 81% of actual positive samples.
- F1 Score: Balances precision and recall into a single metric
$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$
The F1-score is 0.805894, reflecting a balanced trade-off between precision and recall.

Validation Set Performance

The detailed classification report for both classes is as follows:

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.81	0.80	0.80	21197
1 (Positive)	0.80	0.81	0.81	21199
accuracy			0.80	42396
macro avg	0.81	0.80	0.80	42396
weighted avg	0.81	0.80	0.80	42396

Table 2: Classification Report

3.4.1. ROC curve. Receiver Operating Characteristic (ROC) Curve

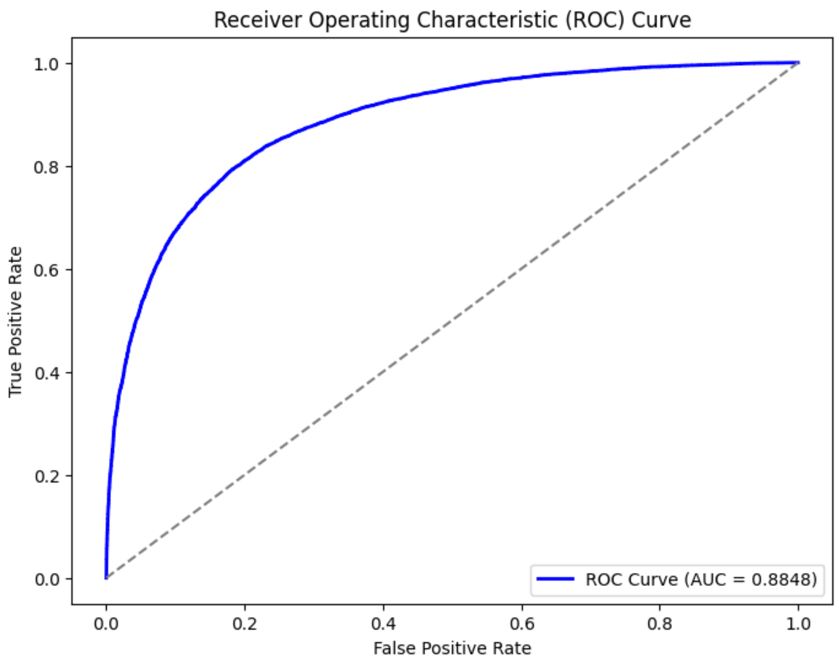

Figure 5: Receiver Operating Characteristic (ROC) Curve

3.4.2. *Learning Curve.* Learning Curve for Logistic Regression

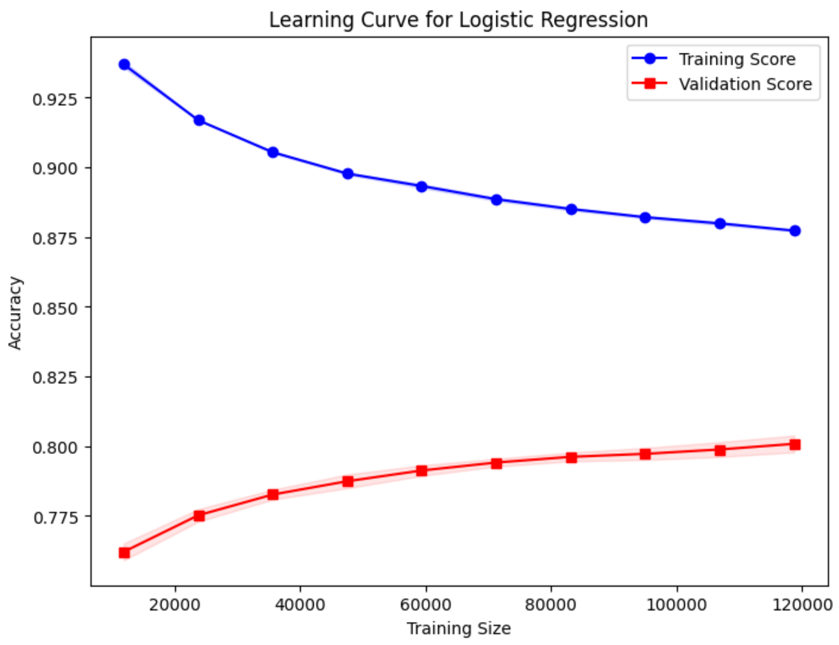

Figure 6: Learning Curve for Logistic Regression

3.4.3. *Confusion matrix.* Confusion matrix

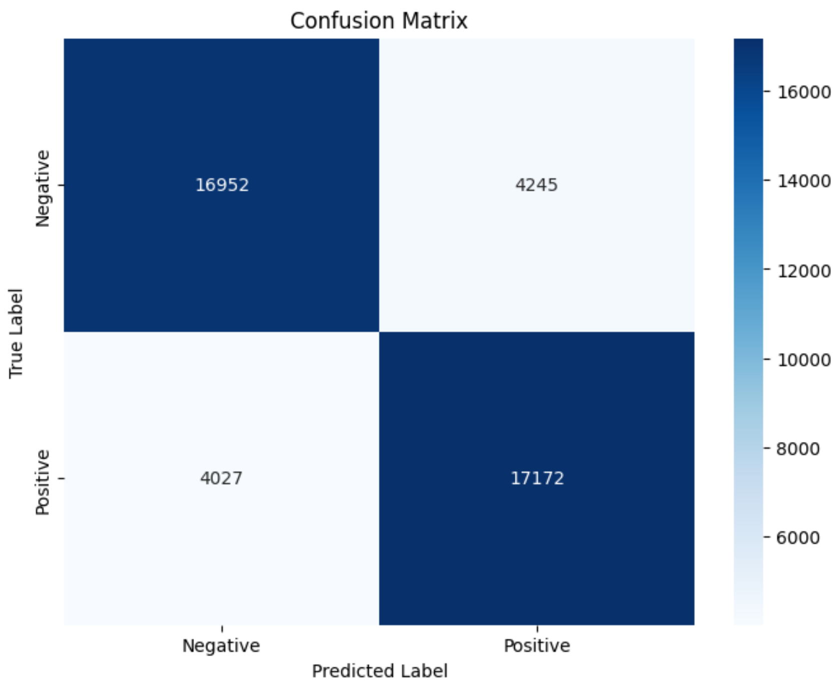

Figure 7: Confusion matrix

4. Results and Overall Analysis

4.1. Results Analysis

For a baseline model using Logistic Regression and TF-IDF features, a validation accuracy of 80.72% is considered good. It indicates that the model is able to generalize well to unseen data and capture the underlying patterns in the text. The balanced precision, recall, and F1-scores suggest that the model is not biased towards one class, which is important for sentiment analysis tasks. Given the simplicity of the model (Logistic Regression with TF-IDF), this performance is in line with expectations. More advanced models (e.g., deep learning or ensemble methods) could potentially achieve higher accuracy, but this is a strong starting point.

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 previously mentioned and further error analysis.

4.1.1. Best trial.

- Performance Metrics
  - Validation Accuracy: 80.49%
  - Precision: 80.18%
  - Recall: 81%
  - F1-Score: 80.59%
- Model

  - Sentiment classifier using Logistic Regression and TF-IDF
- Best Hyperparameters
  - C (Regularization Strength): 2
  - Solver: lbfgs
  - Penalty: l2
- Preprocessing
  - Text Cleaning
  - TF-IDF Vectorization

5. Bibliography

References

[1] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. January 12, 2025.

[2] Manolis Koubarakis. Introductory concepts of machine learning, 2025.

[3] Manolis Koubarakis. Regression, 2025.

[4] Yorgos Pantis. Tutorial 1, 2025.

[5] Josh Starmer. Statquest: Logistic regression, 2018.

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