UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: <Βασιλική Καλαντζή>
sdi: <sdi2200057>

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

Contents

1. Abstract

This project focuses on developing a sentiment analysis classifier for English tweets using a minimalistic approach with only Logistic Regression and TF-IDF vectorization. The constrained technical approach (solely using Logistic Regression + TF-IDF) provides a focused framework for examining fundamental NLP classification techniques while maintaining computational efficiency.

2. Data processing and analysis

2.1. Exploratory Data Analysis

In this section, an Exploratory Data Analysis (EDA) will be conducted on the three predefined datasets: train, validation, and test. These datasets have already been split, with the training set used for model training, the validation set for model selection and hyperparameter tuning, and the test set for evaluating the model's generalization to unseen data.

The primary goal of this analysis is to explore and understand the data, searching for possible patterns, gaps, or imbalances which may influence the further processing stages. Specifically, the analysis will focus on:

- The structure and data types of the datasets.
- Missing values and how they are handled.
- The distribution of the target variable (Label).
- Common patterns in the text data (Text).
- Relationships between features.
- The presence of outliers and their potential impact.

These insights will be critical in developing a robust data preprocessing and enhancement strategy before attempting any modeling or analysis.

2.1.1. Dataset Overview. The first rows of the dataset indicate that each record consists of three columns:

- **ID**: A unique identifier for each tweet.
- **Text**: The content of the tweet.
- **Label**: The sentiment of the tweet, where 1 represents positive sentiment and 0 represents negative sentiment.

The datasets contain the following:

- **Training Set**: Includes 148,388 records with 3 features (ID, Text, Label).
- **Validation Set**: Includes 42,396 records with 3 features (ID, Text, Label).
- **Test Set**: Includes 21,199 records with 3 features (ID, Text, Label).

Additionally, all features have complete values, as **no missing values** were detected in any of the datasets.

2.1.2. Label Distribution. The distribution of labels across the datasets shows an almost perfect balance between the two categories (0 and 1):

Training Set

- **Label 0**: 74,192 samples.
- **Label 1**: 74,196 samples.

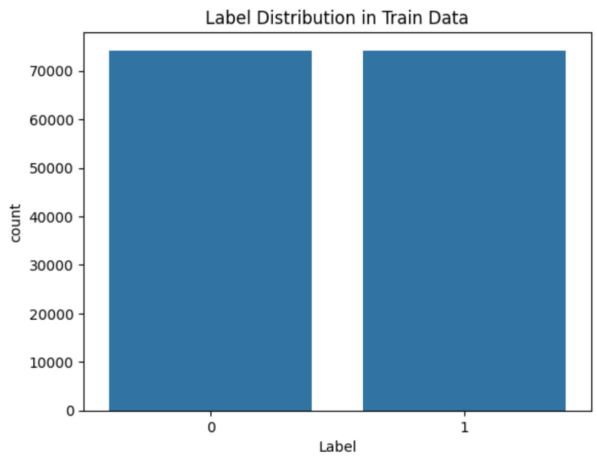

Figure 1:

Validation Set

The distribution is similar to the training set, indicating that the data is well-balanced between the two categories.

- **Label 0**: 21197 samples.
- **Label 1**: 21199 samples.

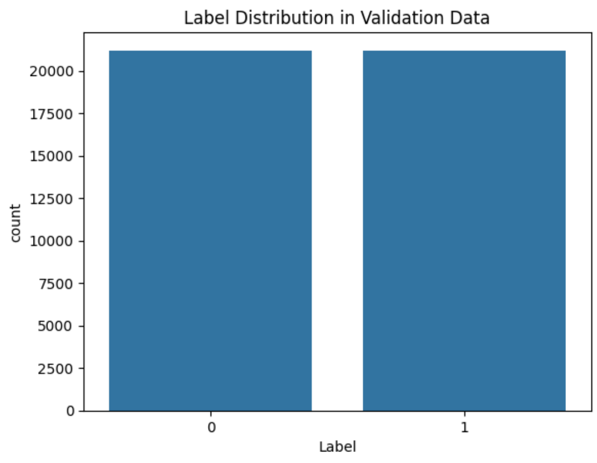

Figure 2:

This balance is crucial as it ensures that the model does not develop a bias toward one of the two categories.

2.1.3. Unique Words. The number of unique texts indicates the presence of overlap and potentially similar patterns across the datasets:

- **Training Set**: 145,608 unique texts out of 148,388 total texts. This implies that there are 2,780 duplicate texts.
- **Validation Set**: 41,788 unique texts out of 42,396 total texts, meaning there are 608 duplicate texts.
- **Test Set**: 20,965 unique texts out of 21,199 total texts, meaning there are 234 duplicate texts.

The presence of duplicate texts can lead to **overfitting** during model training, as the model may memorize repeated texts instead of learning to generalize effectively.

The variation in text length across datasets could impact the model's performance. Below is a breakdown of the unique text lengths observed in each dataset:

- **Training Set**: 210 unique text lengths, ranging from 1 to 210 words.
- **Validation Set**: 162 unique text lengths, ranging from 1 to 162 words.
- **Test Set**: 152 unique text lengths, ranging from 1 to 152 words.

Since the texts in the test dataset are notably shorter than those in the training dataset, the model may have difficulties in generalization. During training, the model learns primarily from longer sequences, which means that it may develop a bias toward processing and understanding them better. During the test phase, when the model is presented with shorter texts, it might not extract enough contextual information, leading to lower accuracy. This discrepancy can impact performance, especially when the model is expected to perform tasks highly dependent on context.

All datasets contain **2 unique labels** (0 and 1), corresponding to negative and positive sentiment, respectively. This indicates that the datasets are **balanced**, which is important to train the model.

2.1.4. Statistical Overview of Label Distribution. In this section, we present a statistical analysis of the distribution of labels in the train and validation data sets. The following table includes basic statistical measures for the **Label** column in both data sets.

The table presents the basic statistical measures for the Label column in the train and validation datasets. The data show that both the train dataset and the validation dataset are balanced, with a mean value very close to 0.5. This means that the two categories (0 and 1) have approximately the same number of records. The standard deviation (std) is very close to 0.5, which is expected for a binomial distribution (0 and 1). Finally, the labels take only the values 0 and 1, as expected, and there are no outliers or invalid values.

The balance in the datasets is very important for training the model, as it ensures that the model will not develop a bias towards one of the two categories. This balance

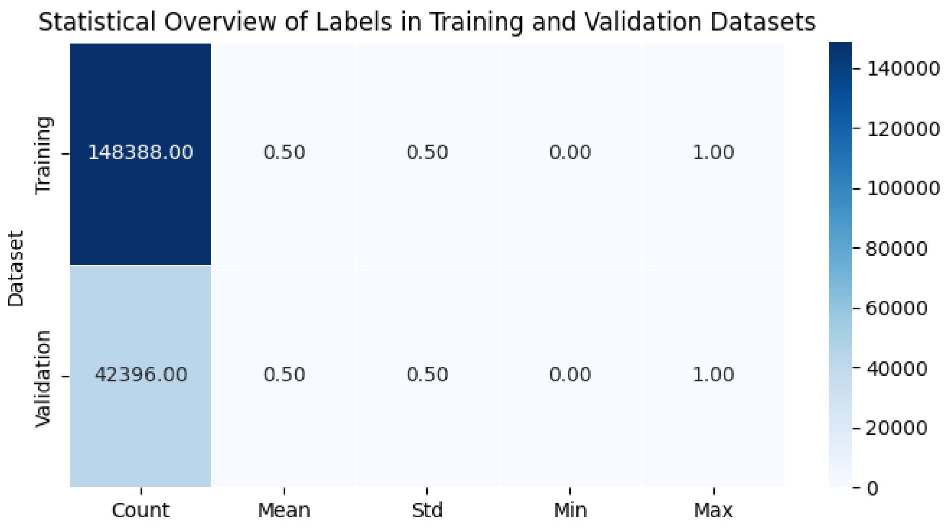

Figure 3:

is also evident in the **label distribution**, where the two categories (0 and 1) have approximately the same number of records.

Furthermore, the consistency between the datasets (train and validation) indicates that the validation dataset is **representative** of the train dataset, which is essential for reliable model evaluation.

Finally, the absence of outliers or invalid values in the **Label** column confirms that the data are clean and ready for further analysis and modeling.

2.1.5. Word Cloud. In the Word Cloud representation, the size of each word indicates its frequency in the training dataset, meaning the larger the word, the more frequently it appear. So, Words such as "now", "love", "quot", "work", "got", "today", "thank", "going", "good", "miss", and "sad" are highly common due to the expected themes of daily life emotions, as expected in a dataset derived from tweets. The vocabulary reveals a balance between positive and negative sentiments, for example words like "love", "thank", and "good" express positive emotions, while "miss", "sad", and "bad" suggest negative emotions. The presence of these categories shows us the diversity of the data which is useful for sentiment analysis model.

Additionally, the presence of slang words and informal expressions, such as "gonna", "wanna", "lol", "nah", and "omg", is observed, which are common in informal online communication. These words make careful preprocessing essential, as they may either need to be preserved to maintain the emotional tone of the texts or replaced with equivalent words (e.g., "gonna" $\rightarrow$ "going to") for better classification by the model.

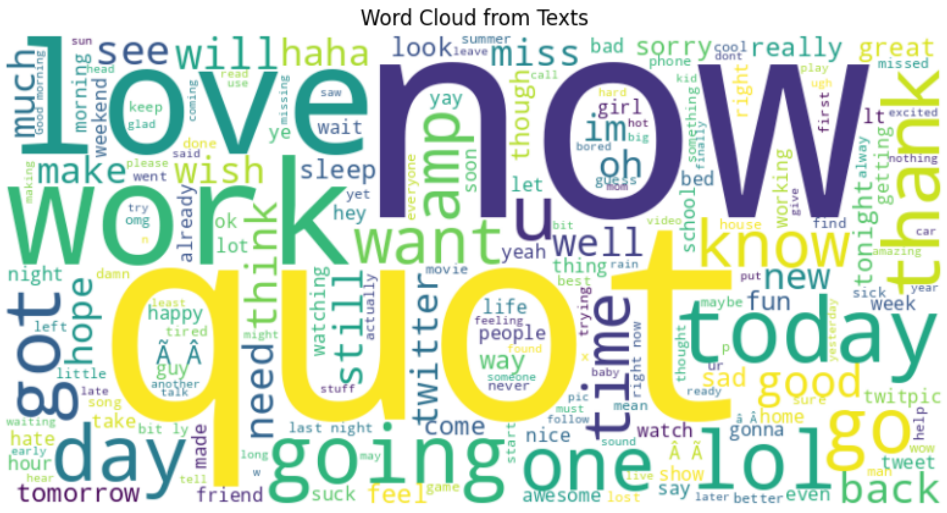

Figure 4:

2.1.6. Histogram of Word Frequencies. The Histogram of Word Frequencies is more suitable for statistical analysis and preprocessing improvement, in contrast to the Word Cloud, which is useful for detecting thematic patterns. This is because, by default, the WordCloud library removes common stopwords such as "the", "to", and "I". The histogram, on the other hand, displays the most frequent words in the dataset, including stopwords, providing a more accurate picture of the word distribution.

This feature makes the histogram particularly useful for identifying stopwords that should be removed during preprocessing, especially if they are not useful for sentiment analysis or for detecting lexical anomalies, such as overly frequent abbreviations or characters that have not been properly removed. Therefore, the histogram contributes to the optimization of preprocessing, enabling more targeted interventions in data cleaning and transformation.

2.1.7. Boxplot- Text length per Label. The boxplot below shows the text length distribution per Label (Label 0 represents negative sentiment, and Label 1 represents positive sentiment) in the training dataset, which consists of tweets. We observe that the distribution of tweet length is similar for both categories, as the majority of tweets have a length between 30 and 120 characters, with a median of approximately 60 characters. The presence of individual tweets with significantly longer lengths (160+ characters) is considered outliers, but their frequency is not particularly high. These tweets do not need to be removed, as they are not problematic and may contain useful information. The similar distribution of length between the two labels is particularly positive for the model that will be trained later, as it indicates that there is no bias regarding the length of tweets across different sentiment categories, so the model will not be able to predict sentiment based solely on text length. This specific analysis can assist in preprocessing, as it may lead to decisions such as removing very short tweets (<10 characters) that may not provide sufficient content.

2.1.8. Bigrams - Trigrams. It is noted from the most frequent bigrams and trigrams shown below, that tweets tend to refer to places or destinations ("in the," "on the," "go to the"), express intentions or desires ("want to," "going to," "looking forward to"),

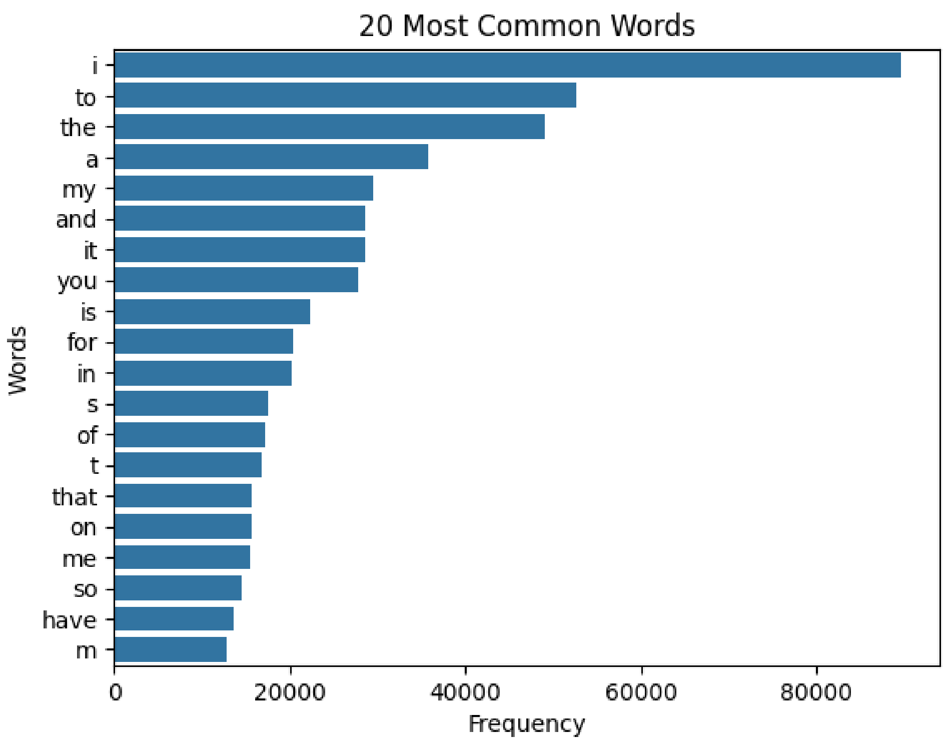

Figure 5:

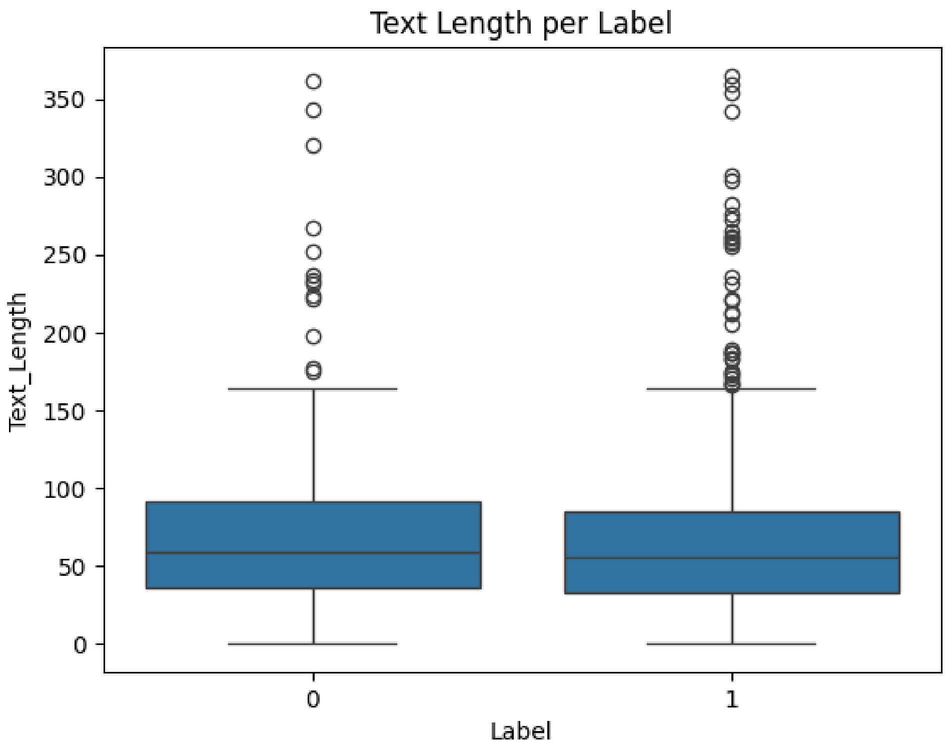

Figure 6:

and include links ("http twitpic com," "http bit ly"). Some of these phrases convey emotions (e.g., "looking forward to" suggests a positive sentiment, while "don't want to" may indicate a negative sentiment), while others are neutral and belong to stopwords (e.g., "in the," "to be"). This distinction is critical to data preprocessing because stopwords, neutral sentences and links can be removed to reduce noise, but caution should be used, as some sentences hold valuable information about emotions or context. Therefore, carefully managing these phrases during preprocessing can enhance sentiment analysis without losing valuable information that enhances tweet content understanding.

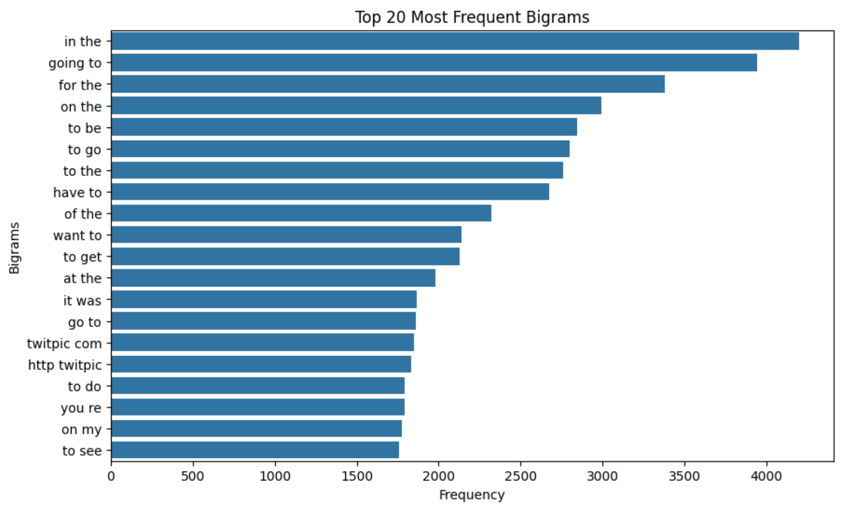

Figure 7:

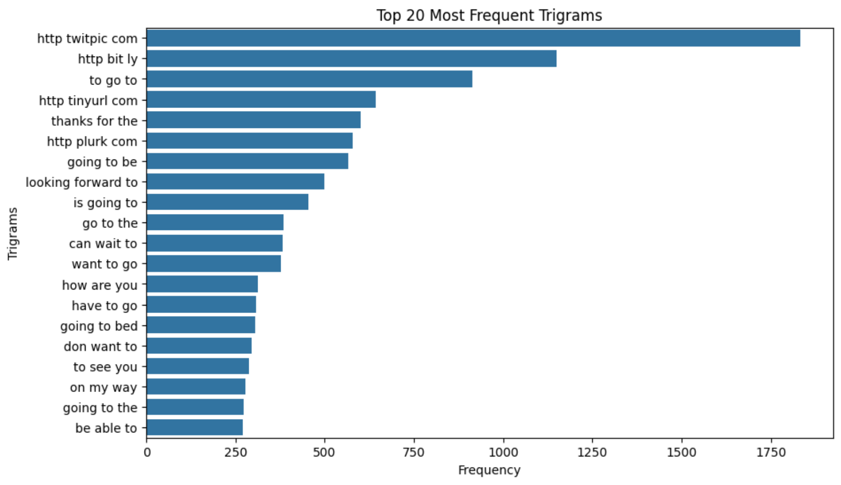

Figure 8:

2.2. Pre-processing

The dataset consists of tweets which often include informal language, slang, hashtags, mentions, links and special characters, making pre-processing an essential step.

Each dataset is unique, so not all pre-processing techniques are universally beneficial. Therefore, it is essential to experiment with different pre-processing methods to identify the most effective approach based on evaluation metrics and the potential for overfitting. Below is a summary of the techniques that were tested and retained or discarded based on their contribution to key evaluation metrics (e.g., accuracy, precision, recall, F1 score) and their effect on overfitting.

2.2.1. Effective Preprocessing Techniques.

- **Removing URLs, Mentions, Numbers and Special Characters**
  This is a highly effective method, since tweets are filled with links, mentions, numbers and special characters. Specifically, URLs and mentions are eliminated to prevent models from learning irrelevant patterns, and numbers are removed to maintain textual consistency and reduce sparsity. The impact of this preprocessing step on model performance is demonstrated in the table below.

Metric	**Value**
Training Accuracy	0.82
Validation Accuracy	0.79019
Precision	0.78710
Recall	0.79560
F1-Score	0.79132
Overfitting	3%

Table 1: Performance Metrics

- **Converting text to Lowercase**
  Lowercasing was applied to make the text consistent by converting all characters to lowercase. This technique is commonly used to ensure that the model does not treat the same word with different capitalizations (e.g., 'Data' vs 'data') as distinct entities.

- **Unicode Character Removal**
  Tweets often contain special Unicode characters that do not contribute to sentiment analysis (for example :
  $$\tilde{A} \quad \hat{A}$$
). Removing them ensures data consistency and prevents unnecessary noise in the model. This step helped in normalizing text and preventing unexpected tokenization errors. The model performed better with clean and standardized input, improving both accuracy and F1-score.

- **Emoticon Replacement with Text Tokens** Tweets often contain emoticons, which provide important emotional cues. Instead of removing them, they were replaced with corresponding text tokens that preserve their sentiment. Specifically, a predefined mapping of common emoticons to sentiment-based text tokens (e.g., :) = [happy], :(= [sad]) was used and each emoticon in the text was replaced accordingly. This step was retained as it preserves emotional context, which is critical for sentiment analysis tasks. Instead of treating emoticons as noise, the model can leverage them as meaningful sentiment indicators, improving classification performance.

- **Social Media Sentiment Substitutions**
  Tweets frequently includes expressions such as "lol," "omg," or "smh," which convey emotions or reactions. So, a set of regex-based substitutions was applied to detect common social media expressions and replace them with their sentiment-based counterparts. For example, "lol" and "hahaha" were replaced with [happy], while "ugh" was replaced with [annoyed]. This preprocessing step helped retain sentiment information that would have been lost if these expressions were simply removed. It improved recall in detecting sentiment-heavy tweets without significantly affecting overfitting.

- **Negation Handling** Negation in a sentence can completely alter its meaning, directly impacting the accuracy of an NLP model. If the model does not properly process negation, there is a risk of incorrect sentiment classification (e.g., "I am happy" conveys a positive sentiment, whereas "I am not happy" has the opposite meaning). To address this issue, a technique is applied that detects negation words such as not, never, don't, and can't, converting phrases like not happy into not_happy. This approach significantly improved the model's accuracy, as it more effectively captured negative sentiments and reduced classification errors.

2.2.2. Ineffective Preprocessing Techniques.

- **Tokenization-Lemmatization**
  Tokenization is the process of splitting a text into individual words or tokens. This helps the model recognize words as separate units and process them correctly. Lemmatization then reduces words to their base form, ensuring that different variations of the same word are treated as one. This simplifies the data and improves model performance. The effectiveness of lemmatization is enhanced by lowercasing, as it ensures that words are comparable and consistent. By converting all text to lowercase, lowercasing allows lemmatization to work more effectively, reducing unnecessary morphological variations and enabling the model to better understand the meaning of words. Although lemmatization slightly reduced accuracy, it improved the F1-score, which is a more important performance metric as it balances both precision and recall. This is particularly useful when dealing with imbalanced datasets. This is the reason why I retained the tokenization and lemmatization preprocessing techniques. When combined with the removal of URLs, mentions, numbers, and special characters, the final performance metrics of my preprocessing function are shown below.

- **Stemming**
  Stemming is a text preprocessing technique that reduces words to their root or base form by removing suffixes (e.g., "running" becomes "run"). Unlike lemmatization, which considers the grammatical structure and meaning of words, stemming applies a more aggressive approach, often cutting words down to a common substring. When tested, stemming led to a decline in all evaluation metrics. This drop is likely because stemming can over-simplify words, stripping away

Metric	**Value**
Training Accuracy	0.82
Validation Accuracy	0.79014
Precision	0.78678
Recall	0.79602
F1-Score	0.79138
Overfitting	3%

Table 2: Final Performance Metrics

meaningful suffixes and reducing clarity. For example, it might convert "university" and "universal" to the same root ("univers"), making it harder for the model to distinguish between them. Given its negative impact on evaluation metrics, stemming was excluded from the final preprocessing function.

Metric	**Value**
Training Accuracy	0.82
Validation Accuracy	0.78573
Precision	0.78246
Recall	0.79154
F1-Score	0.78698
Overfitting	3%

Table 3: .

- **Correction of Spelling Errors**
  The manual correction of spelling errors and slang terms similarly demonstrated adverse effects on model performance, as evidenced by reduced evaluation scores. Substitutions like "u" $\rightarrow$ "you" and "luv" $\rightarrow$ "love" likely introduced noise through erroneous modifications, particularly when abbreviations carried context-specific meanings (e.g., "U" as a university acronym). Furthermore, automated spelling correction libraries (SymSpell, TextBlob) not only decreased accuracy but also proved computationally expensive for large datasets, making them impractical for our pipeline.

- **Removal of Stopwords**
  Stopwords are commonly removed in text pre-processing to reduce noise and focus on meaningful words. However, in this case, the removal of stopwords reduced all metrics and increased overfitting. This is likely because some stopwords, particularly negation words (e.g., "not," "no," "don't"), carry important semantic information, especially in tasks like sentiment analysis. For example, the presence of negation words can completely change the meaning of a sentence (e.g., "I am happy" vs. "I am not happy").

  To test this hypothesis, the pre-processing was modified to retain negation words while removing other stopwords. However, even with this adjustment, the performance metrics did not improve, as shown in the table below. This suggests that the removal of stopwords, in general, is not beneficial for this specific dataset and task. Therefore, stopwords were retained in the final pre-processing function.

Metric	**Value**
Training Accuracy	0.82
Validation Accuracy	0.78
Precision	0.7771
Recall	0.7941
F1-Score	0.7855
Overfitting	4%

Table 4: Performance Metrics After Removing Stopwords

- **Part of Speech (POS) Tagging for Noun and Verb Retention**
  Part-of-speech (POS) tagging was explored as a pre-processing step, where only nouns (NN) and verbs (VB) were kept in the text. The idea behind this approach was to focus on the most essential parts of speech, as nouns and verbs often carry key information about a sentence's meaning. Other word types, such as adjectives, adverbs, and prepositions, were excluded under the assumption that they might contribute less to the model's understanding of the text. However, this technique led to a significant reduction in all evaluation metrics and overfitting increased, suggesting that the model became less generalizable. This indicates that removing adjectives, adverbs, and other grammatical components resulted in the loss of valuable context, making it harder for the model to make accurate predictions.

Metric	**Value**
Training Accuracy	0.79
Validation Accuracy	0.7355
Precision	0.7286
Recall	0.7508
F1-Score	0.7395
Overfitting	5%

Table 5: Performance Metrics After Retaining Only Nouns and Verbs

2.3. Exploratory Data Analysis After Preprocessing

From the first 5 lines of the dataset before and after preprocessing, it appears that:

- **Conversion of Mentions (@username)**
  Before preprocessing, user mentions (@MadeleineBCN) appeared with the @ symbol. After preprocessing, the @ symbol was removed while keeping the username, which was converted to lowercase (madeleinebcn). This change helps standardize the text while preserving user information that may be important for specific analyses.

- **Lowercase Conversion**
  All text was converted to lowercase, as shown in the example Their → their.
- **Simplification of Repeated Characters**
  Repeated characters (multiple dots like ...) were simplified to two dots (..).
- **Removal of Special Characters and Numbers**
  Special characters ("-") were removed, converting words like *i-Top* to *itop*. Additionally, numbers were eliminated from the text.

Increased text length after preprocessing (shown by the higher orange boxplot) results from intentional design choices that preserve information:

- Adding interpretative tokens (e.g., converting `:)` `→` `[happy]` or marking negations with `[NEG]`)
- Retaining usernames (without `@`)
- Deliberately keeping stopwords to maintain natural flow and completeness.

While these decisions increase text length, they significantly improve linguistic analysis accuracy and ensure meaning preservation. The added tokens and maintained elements provide crucial contextual information that would otherwise be lost in more aggressive preprocessing.

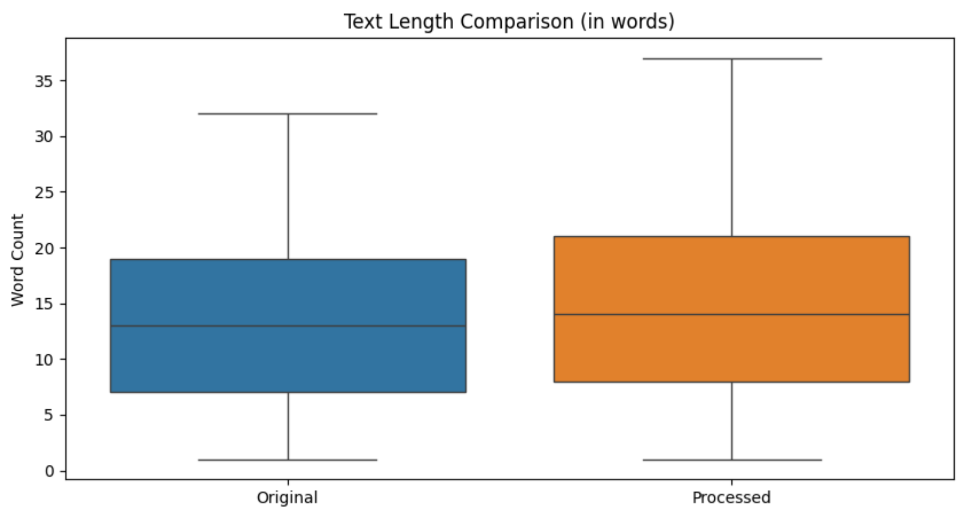

Figure 9:

From the word cloud, it is evident that one of the most frequent words is NEG, which is expected due to the negation handling performed during preprocessing. Additionally, it becomes clear, as before, that punctuation is absent from words like dont and am, further confirming that stopwords have not been removed. Moreover, Unicode characters are no longer visible, unlike in the word cloud before preprocessing (for example :

$$\tilde{A} \quad \hat{A}$$

).

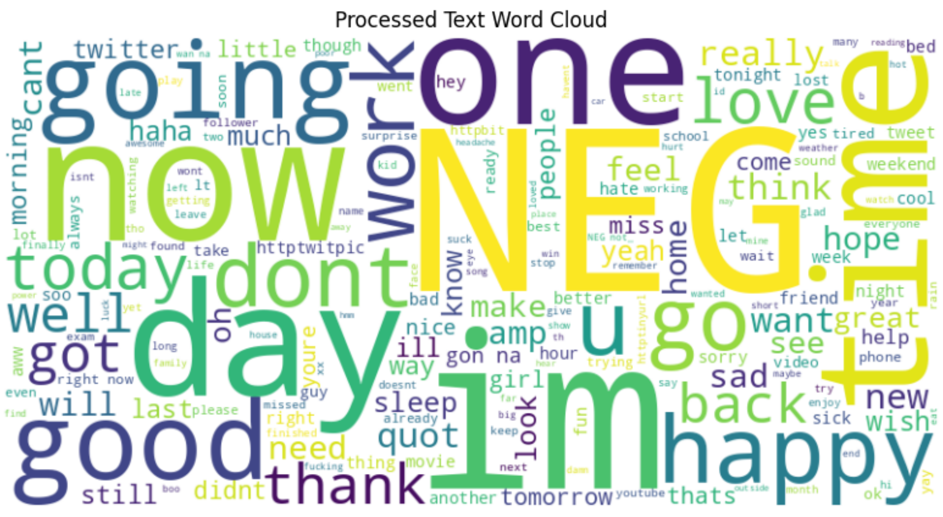

Figure 10:

The analysis of the 20 most frequent words, bigrams, and trigrams confirms that URLs have been removed, as terms such as https and com were among the most common n-grams before preprocessing. Additionally, it becomes evident that slang words, such as gonna, have not been removed, in accordance with the design choices outlined in the preprocessing process. Finally, the retention of stopwords is reaffirmed, contributing to the preservation of the natural flow and overall completeness of the text.

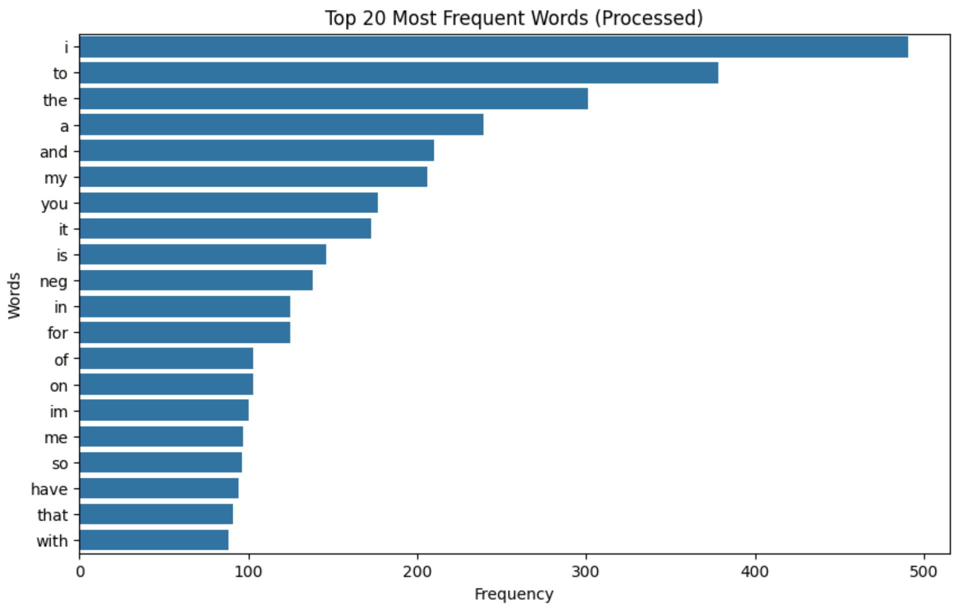

Figure 11:

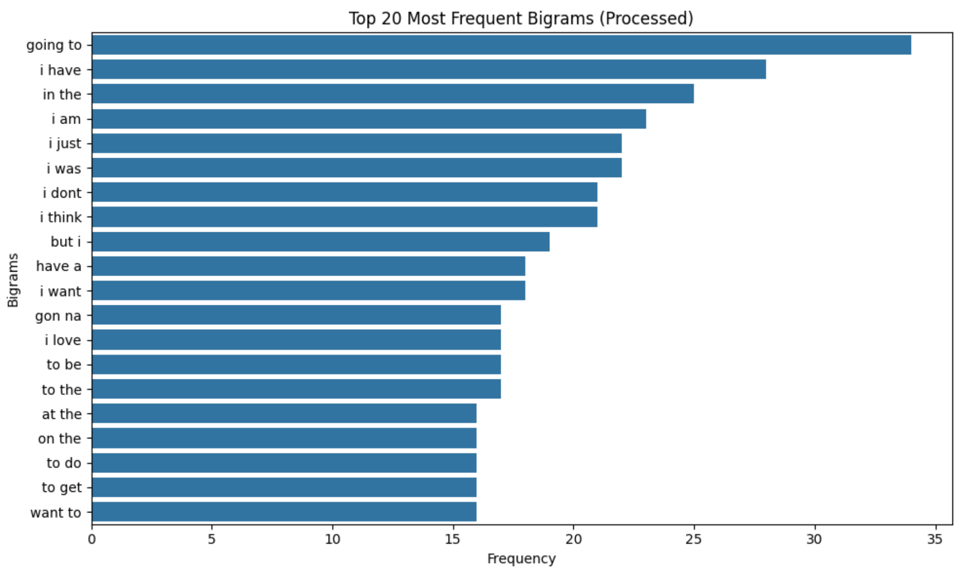

Figure 12:

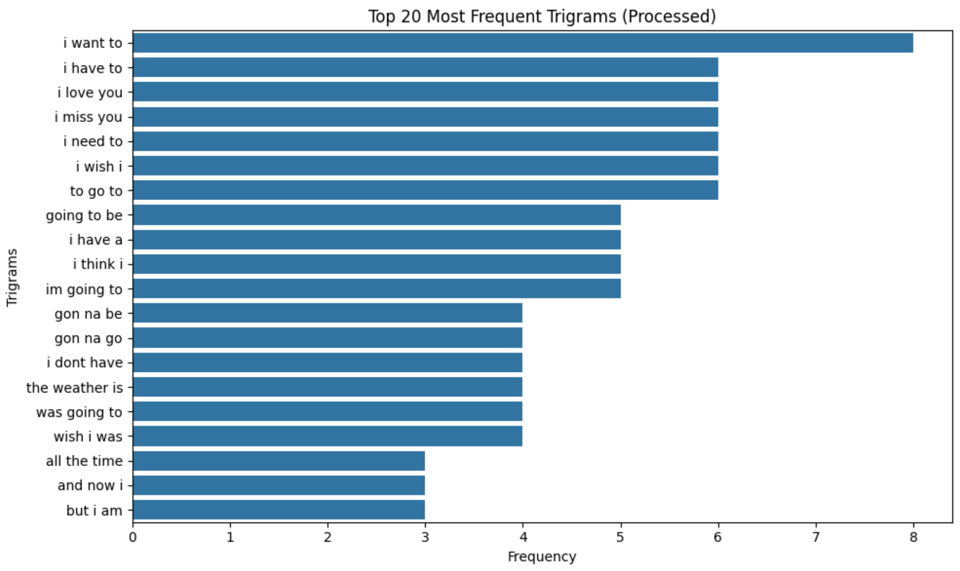

Figure 13:

3. Algorithms and Experiments

3.1. Experiments

For the development of the sentiment classifier, Logistic Regression was used in combination with TF-IDF to transform text into numerical features. Initially, a simple model was implemented without extensive optimization to establish a baseline for further experimentation.

To improve the results, we conducted various experiments with different hyperparameter settings. Different values for the **regularization strength (C)** were tested to achieve the optimal balance between bias and variance. Additionally, we explored various configurations for **ngram_range, max_df, and min_df** to identify the most relevant words or phrases and were tested different Logistic Regression solvers, such as `liblinear` and `saga`, to determine the most efficient training algorithm.

After text preprocessing, the initial model was evaluated using the following metrics:

Metric	Value
Training Accuracy	0.83460
Validation Accuracy	0.79113
Precision	0.78873
Recall	0.79532
F1-Score	0.79201
Overfitting	4%

Table 6: Performance metrics of the initial model

The model demonstrated decent performance; however, a slight tendency for **overfitting** was observed, as indicated by the **4% gap** between Training and Validation Accuracy.

3.2. Hyper-parameter Tuning

Hyperparameter optimization was initially performed using **Grid Search** with **5-Fold Cross Validation**.

Grid Search is an exhaustive process that searches for the optimal values of the model's hyperparameters by testing multiple combinations and selecting the best based on validation performance.
5-Fold Cross Validation means that the dataset is split into five equal parts. The model is trained five times, each time using **four of the five parts** as the training set and the remaining one as the validation set. In the end, the overall performance is computed as the **average of the five iterations**. This method helps mitigate overfitting and provides a more reliable estimate of the model's general performance.

However, since **Grid Search** can be computationally expensive, especially for large datasets, we also tested **Cross Validation with 3 folds**, combined with a more targeted selection of parameters to speed up the process.

3.2.1. Examined Hyperparameters. The key hyperparameters examined include:

- `tfidf__ngram_range`
- `tfidf__min_df & tfidf__max_df`
- `model__C` (Regularization Strength)
- `model__solver` (Optimization Algorithm)
- `model__class_weight` (Handling Imbalance)
- `model__penalty` (Regularization Type, e.g., L1, L2)

3.3. TF-IDF parameters

For the transformation of texts into TF-IDF vectors, several experiments were conducted while keeping the parameters of the Logistic Regression fixed (it was later realized that with different solvers, there was no need to change the TF-IDF parameters as they did not affect the evaluation metrics, except for a few exceptions), with the aim of extracting optimal parameters that balance accuracy, generalization, and computational efficiency. Specifically, broad ranges of values for each hyperparameter were tested, based on the literature and empirical observations. All parameters were then evaluated using 5-fold cross-validation for reliable estimation.

Parameter	**Optimal Value**	**Tested Range**	**Purpose**	**Performance Impact**
min_df	8	5-15	Filters terms appearing in fewer than N documents (noise reduction)	Setting min_df=8 improved accuracy by removing rare, potentially noisy terms and prevents overfitting by excluding terms with insufficient representation.
max_df	0.3	0.2-1.0	Excludes terms appearing in >X% documents (common/irrelevant words)	Improved accuracy by filtering out overly frequent, less meaningful words (e.g., stopwords) and helps focus on more discriminative terms.
ngram_range	(1,2) or (1,3)*	(1,1), (1,2), (1,3)	Controls single words (unigrams) vs. word combinations (bigrams/trigrams)	Bigrams/trigrams improved accuracy by capturing context but increased computational cost and memory usage due to higher feature dimensionality.
sublinear_tf	True	True/False	Applies logarithmic TF scaling	Improved accuracy by reducing the dominance of high-frequency terms, with minimal computational overhead, making it an efficient optimization

3.4. Logistic regression parameters

After experimenting with TF-IDF parameters, we proceeded to test Logistic Regression configurations.

- **Penalty = 'l2'**
  L2 regularization applies smooth shrinkage to weights, preserves all features (unlike L1) and achieved better accuracy in combination with L-BFGS. However, after experimentation it was observed that L1 regularization is more effective, when combined with the Liblinear solver. L1 regularization performs feature selection by zeroing out insignificant coefficients, retaining only the critical features and in Liblinear, it is applied with optimized algorithms.

- **C = 1.0 (Regularization Intensity)**
  The parameter C controls the balance between Overfitting (small C = excessive regularization) and Underfitting (large C = excessive flexibility).

- **solver = L-BFGS**
  Experiments were conducted with multiple solvers, including saga, sag, and liblinear. While liblinear is optimized for small datasets and saga performs well with large-scale sparse data, lbfgs consistently yielded the highest accuracy in our experiments. Given that our dataset is large, lbfgs's ability to efficiently handle the complexity of the data proved particularly important. This is because lbfgs effectively approximates the Hessian matrix, enabling better optimization of the loss function, especially in high-dimensional feature spaces like those produced by TF-IDF. Its ability to smoothly manage L2 regularization further contributed to its superior performance in this task.

- **max_iter=300**: The maximum number of iterations is limited to 300. Convergence behavior revealed that L-BFGS typically requires more iterations to reach optimal solutions. This stems from its more sophisticated Hessian approximation calculations, though the tradeoff yields stable convergence without divergence risks.

- **multi_class = 'ovr' (One-vs-Rest)**
  This approach creates a separate binary model for each class. The 'ovr' (One-vs-Rest) strategy was empirically selected over multinomial (softmax) as it yielded higher accuracy in our experiments. This occurred because the classes in our dataset were naturally well-separated, where OVR's binary decomposition proved sufficient, while softmax's added complexity might have introduced unnecessary noise.

- **class_weight = None**
  All classes are given equal weight during computation. The dataset used (tweets) has a balanced class distribution. Trials with class_weight $=$ balanced reduced accuracy by introducing bias towards minority classes without justification.

Trial	Solver	Penalty	C	Train Accuracy	Validation Accuracy
Baseline	-	-	-	0.83460	0.79113
Trial 1	liblinear	l1	1.0	0.824838	0.805005
Trial 2	lbfgs	l2	1.0	0.844570	0.805831
Trial 3	saga	l2	1.0	0.846248	0.805760

Table 8: Trials

4. Results and Overall Analysis

4.1. Results Analysis

The final logistic regression model with TF-IDF vectorization and custom text preprocessing demonstrated robust performance, achieving a test set accuracy of **0.80536** (Kaggle score), consistent with our initial projections. This model configuration was selected after thorough evaluation of the performance-generalization tradeoff.
While the model exhibits approximately 4% overfitting (evident in the marginal divergence between training and validation metrics), it maintains superior evaluation metrics (accuracy, precision, recall, F1-score) compared to alternative approaches. This degree of overfitting was deemed an acceptable compromise given:

- The model's **substantially better predictive performance** relative to other configurations tested
- Its maintained ability to **generalize effectively** to unseen data
- The observation that this limited overfitting has **negligible impact** on practical deployment performance
- The **meaningful improvements** in key evaluation metrics that justified this balanced approach

The selected configuration represents an **optimal balance** between model complexity and generalization capability for this specific application.

Table 9: Optimal Parameters for Final Model Pipeline

Component	**Optimal Parameters**
TF-IDF Vectorizer	min_df = 8
	max_df = 0.3
	ngram_range = (1, 3)
	stop_words = None
	sublinear_tf = True
Logistic Regression	C = 1.0
	penalty = ′l2′
	solver = ′lbfgs′
	max_iter = 300
	multi_class = ′ovr′

Table 10: Training and Validation Metrics

Metric	**Training**	**Validation**
Accuracy	0.8461	0.8058
Precision (Macro)	–	0.8059
Recall (Macro)	–	0.8058
F1-Score (Macro)	–	0.8058

4.2. Model Evaluation Metrics

Accuracy.

- **Definition**: Ratio of correct predictions to total predictions
- **Interpretation**:
  - Training: 84.61% (indicates slight overfitting)
  - Validation: 80.58% (good generalization)
  - **Gap**: 4.03% (acceptable overfitting level)

Precision (Macro).

- **Definition**: $\frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i+FP_i}$
- Shows model′s exactness in positive predictions
- 80.59% indicates low false positive rate across classes

Recall (Macro).

- **Definition**: $\frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i+FN_i}$
- Measures model's completeness in finding positives
- 80.58% suggests good coverage of all classes

F1-Score (Macro).

- **Definition**: $2 \times \frac{Precision \times Recall}{Precision + Recall}$
- Harmonic mean of precision and recall
- 80.58% indicates balanced performance
- Most reliable metric for imbalanced datasets

Learning curve. The graph shows that the model converges, as both curves (training score and cross-validation score) tend to stabilize. However, the 4% gap between them suggests slight overfitting but no divergence.

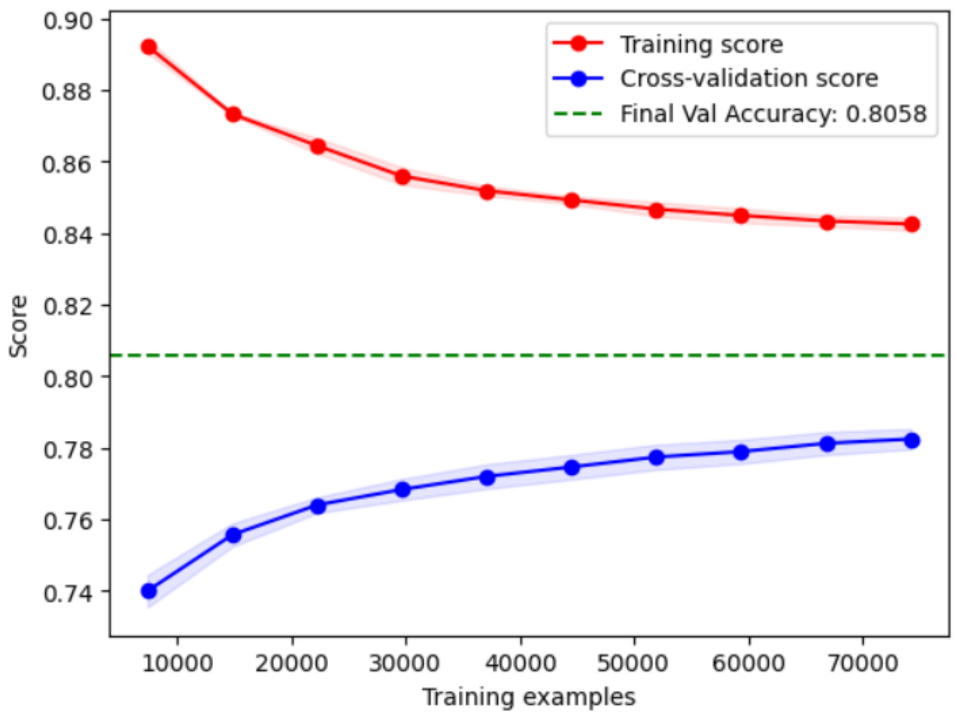

Figure 14:

ROC Curve Analysis

The Receiver Operating Characteristic (ROC) curve graphically represents the classifier's performance across all possible classification thresholds. Specifically:
Axes:

- **True Positive Rate (TPR, sensitivity):** Percentage of correctly classified positive instances.
- **False Positive Rate (FPR):** Percentage of negative instances incorrectly classified as positive.
- **Diagonal Line (AUC = 0.5):** Represents the performance of a random classifier. The ROC curve's position above this line indicates better-than-random performance.

In this specific analysis, the ROC curve shows **AUC = 0.89**, indicating **very good discriminative ability** (close to the ideal 1.0 value). The distance between the curve and the diagonal line confirms that the model effectively separates the classes, with high sensitivity (TPR) and limited false positive rate (FPR). Therefore, the ROC curve analysis confirms that the proposed classifier has strong discriminative capability between classes based on the available features.

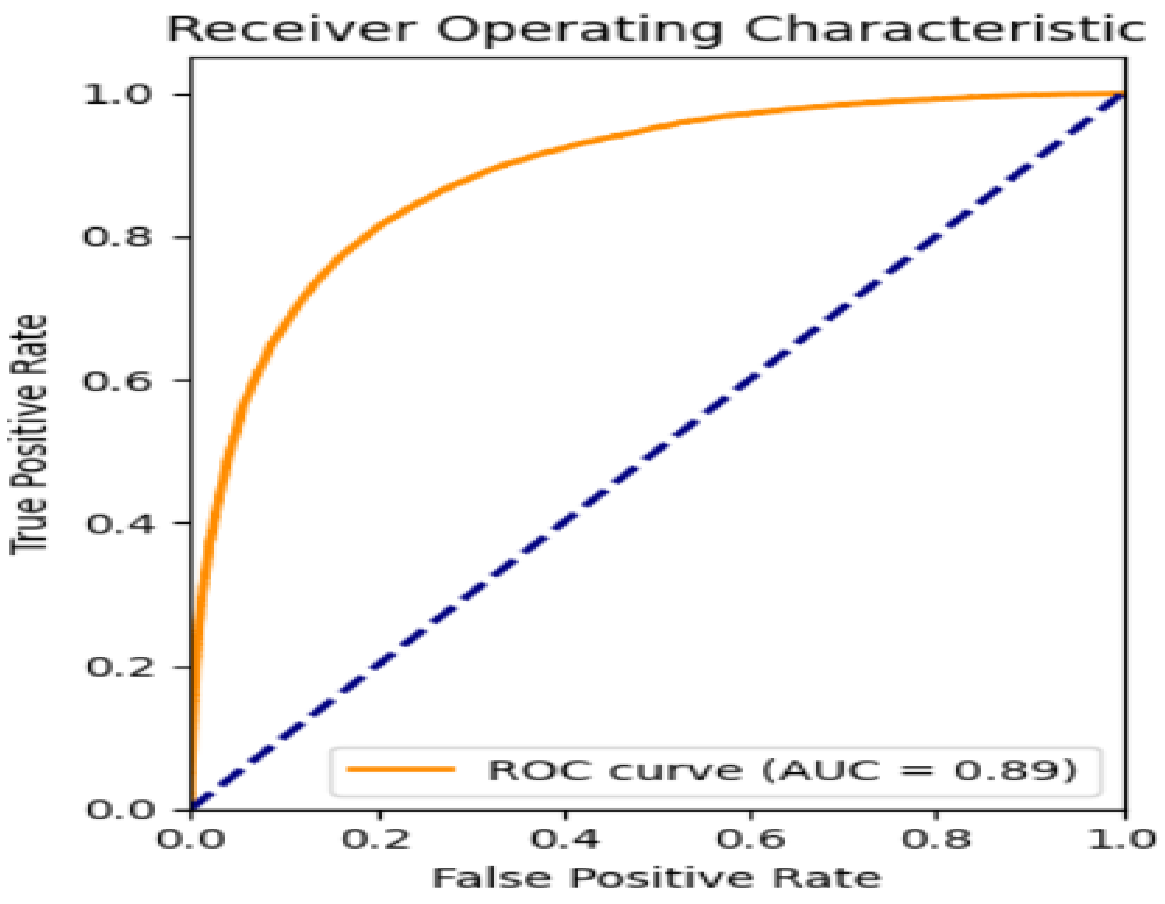

Figure 15: ROC curve (AUC = 0.89)

5. Bibliography

References

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

<Example of citing a source is like this:> [1] <More about bibtex>