UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Artemis Lazanaki*
sdi: <2100081>

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

Contents

1. Abstract

This project focuses on **sentiment analysis**, aiming to classify tweets as **positive** or **negative** based on their content. TF-IDF vectorization is applied to convert text into numerical representations, and Logistic Regression is used as the classification model. Multiple experiments are conducted to optimize model performance using **preprocessing, hyperparameter tuning**, and evaluation metrics such as **accuracy, precision, recall**, and **F1-score**.

2. Data processing and analysis

2.1. Pre-processing

Successful Preprocessing Techniques
To ensure the dataset is clean and suitable for classification, a series of text preprocessing steps were applied. The following transformations were performed:

- **Mojibake and Unicode Fixes:** Non-standard Unicode characters and encoding artifacts were corrected using the `ftfy` library via the function `fix_mojibake(text)`. This step ensured cleaner, human-readable text by repairing encoding issues.
- **URL Replacement:** Instead of removing URLs, they were replaced with the generic token `"url"` to retain sentence structure while preventing excessive noise from unique links.
- **Slang and Abbreviation Expansion:** Informal words and abbreviations frequently used in tweets were replaced with their standard forms using a predefined dictionary. Examples include `"luv"` → `"love"`, `"bc"` → `"because"`, and `"gr8"` → `"great"`.
- **Contraction Expansion:** Contractions were replaced with their full forms to improve textual clarity. For instance, `"dont"` was expanded to `"do not"`, and `"youre"` to `"you are"`.
- **Emoticon and Emoji Replacement:** Common emoticons were replaced with their textual meaning to help capture sentiment. Examples include `":)"` → `"happy"`, `"T_T"` → `"crying"`, and `"<3"` → `"love"`.
- **Repeated Character Reduction:** Exaggerated expressions (e.g., `"sooo happy"`) were normalized by reducing sequences of two or more repeated characters to a double occurrence.
- **Negation Handling:** Negation words such as `"not"`, `"never"`, and `"no"` were identified, and the following word (except some skip words like "too") was combined using an underscore (e.g., `"not happy"` → `"not_happy"`). This transformation helps preserve negation context.
- **Username Anonymization:** Twitter mentions (e.g., `"@user123"`) were replaced with the generic token `"username"` to reduce noise.

- **Lowercasing:** All text was converted to lowercase to maintain uniformity across the dataset.
- **Punctuation Removal or Replacement:** For example, apostrophes were removed before contraction expansion to ensure correct processing.
- **Single Character Word Removal:** Isolated single-character words were removed as they typically do not contribute to sentiment classification.
- **Whitespace Normalization:** Multiple consecutive spaces were reduced to a single space to ensure text consistency.

Alternative Preprocessing Attempts

Several additional preprocessing techniques were tested but resulted in lower model accuracy:

- **Replacing Hashtags with "hashtag":** Initially, all hashtags were replaced with the token `"hashtag"`. However, this approach led to a decline in accuracy, likely because hashtags often contain sentiment-carrying words. Instead, punctuation removal was applied while preserving the individual words within hashtags.
- **Replacing Names with "name":** Common names were initially replaced with the token `"name"`. However, this abstraction removed potential sentiment-related context and negatively affected performance.
- **Replacing Time Expressions with "time" or Keywords:** Time expressions such as `"2:30 PM"` were initially replaced with the token `"hour"` or with descriptive keywords (e.g., `"10 AM"` $\rightarrow$ `"morning"`). These replacements led to a decrease in accuracy. Consequently, time expressions were removed, while the `"AM"` and `"PM"` indicators were retained.

2.2. Analysis

To ensure consistency between the training and validation datasets, both were analyzed.

2.2.1. Class Distribution. The **training dataset** consists of **148,388 tweets**, each labeled as either positive (1) or negative (0). The dataset is nearly perfectly **balanced**, with an almost equal number of positive and negative samples. This balance ensures that the model is not biased toward any particular class, eliminating the need for resampling techniques such as oversampling or undersampling.

Label	Count
Positive (1)	74,196
Negative (0)	74,192

Table 1: Class Distribution in the Training Set

Figure 1: Class Distribution in the Training Set

Similarly, the **validation dataset** consists of **42,396 tweets**, with a nearly identical class distribution, as shown in Table 2.

Label	Count
Positive (1)	21,199
Negative (0)	21,197

Table 2: Class Distribution in the Validation Set

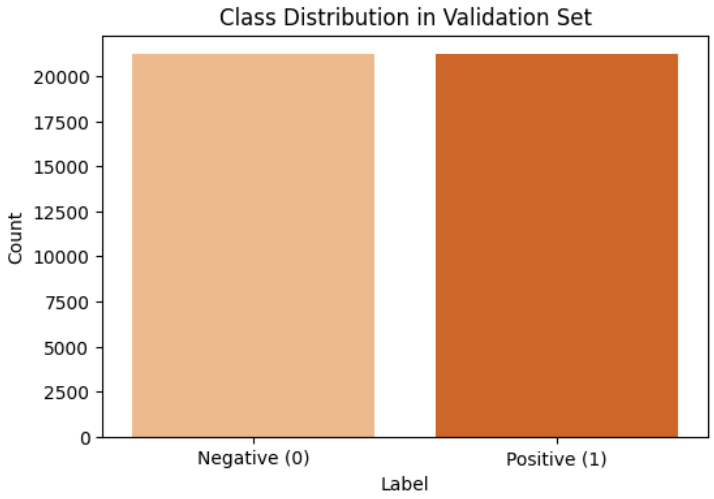

Figure 2: Class Distribution in the Validation Set

The validation set distribution closely mirrors that of the training set, confirming that both datasets have a balanced representation of sentiment classes. This consistency ensures that the validation performance accurately reflects the model's generalization capability.

2.2.2. Tweet Length Analysis. To further investigate dataset characteristics, the **distribution of tweet lengths** (in number of words) was analyzed. Table 3 summarizes the

average tweet length in both datasets.

Dataset	Average Tweet Length (Words)
Training Set	13.29
Validation Set	13.31

Table 3: Average Tweet Length in Training and Validation Sets (Pre-cleaning)

The majority of tweets are relatively short, with a peak around **10–15 words**. The similarity in tweet length distributions across both datasets suggests that the validation set is well-aligned with the training set in terms of textual structure. This consistency ensures the model is trained and evaluated on comparable data.

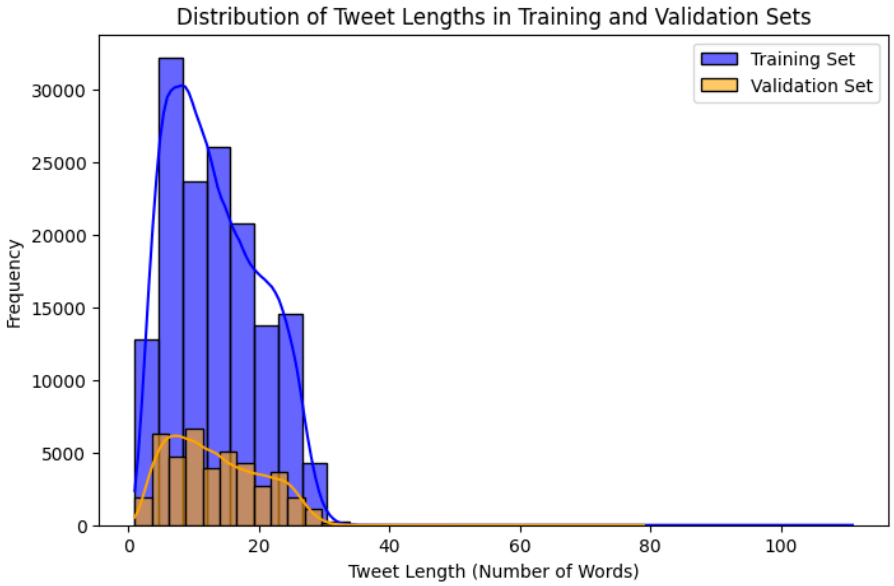

Figure 3: Distribution of Tweet Lengths in Training and Validation Sets (Before Cleaning)

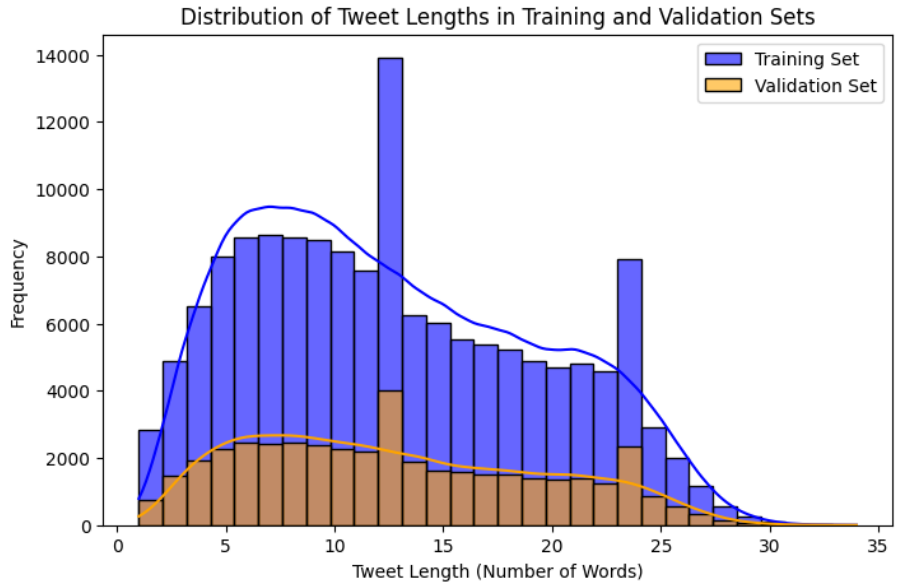

Figure 4: Distribution of Tweet Lengths in Training and Validation Sets (After Cleaning)

Text cleaning led to a reduction in tweet length by approximately **7 words on average**, and a substantial decrease in the number of unique tokens (from 234,492 to 69,936 in training). This suggests that cleaning effectively removed noise and redundancy (e.g., URLs, usernames, punctuation), which can help reduce vocabulary sparsity and improve model generalization.

2.2.3. Word Frequency Analysis. To better understand the dataset and evaluate the impact of preprocessing, word clouds were generated before and after preprocessing for both the training and validation sets.

Key Observations from Word Clouds Before Preprocessing:

- **HTML Artifacts and Encoding Issues:** Words such as `"amp"` (escaped `"&"` symbol) and `"quot"` (escaped quotation mark) appeared frequently, indicating HTML artifacts in the text.
- **Unprocessed URLs:** The presence of terms like `"http"`, `"twitpic"`, and `"bit ly"` suggested that URLs were still affecting token frequencies.
- **Contraction Inconsistencies:** Words like `"im"` frequently appeared in their informal form instead of their expanded equivalent (`"I am"`).

Key Changes from Word Clouds After Preprocessing:

- **User Mentions Replaced:** The generic token `"username"` was prevalent, indicating that a large number of mentions existed in the original dataset.
- **URLs Removed or Replaced:** The token `"url"` appeared, and direct URL fragments like `"bit ly"` were removed.
- **Encoding Artifacts Disappeared:** Unwanted symbols such as `"ÃàÂ"` were cleaned, and HTML artifacts like `"amp"` were removed.

- **Contractions Expanded:** Informal contractions were expanded to their full forms (e.g., `"im"` → `"i am"`, `"ill"` → `"i will"`), making words like `"will"` appear in the word cloud.

The word clouds below illustrate these differences. The figures show the most frequent words in the dataset before and after preprocessing.

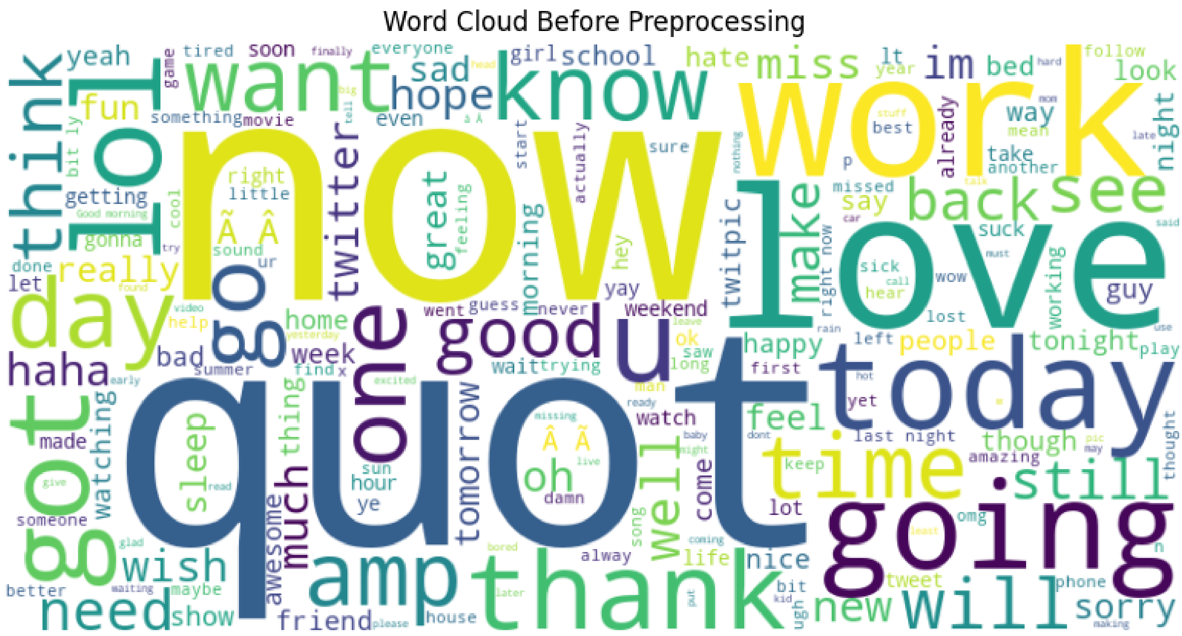

Figure 5: Word Cloud Before Preprocessing (Training Set)

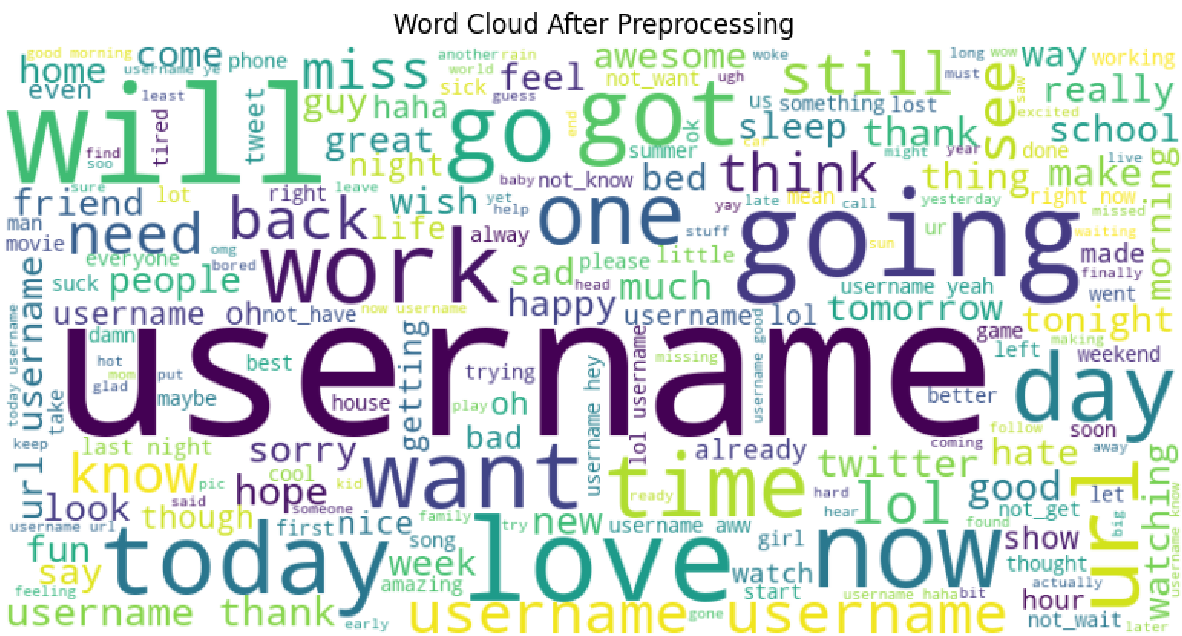

Figure 6: Word Cloud After Preprocessing (Training Set)

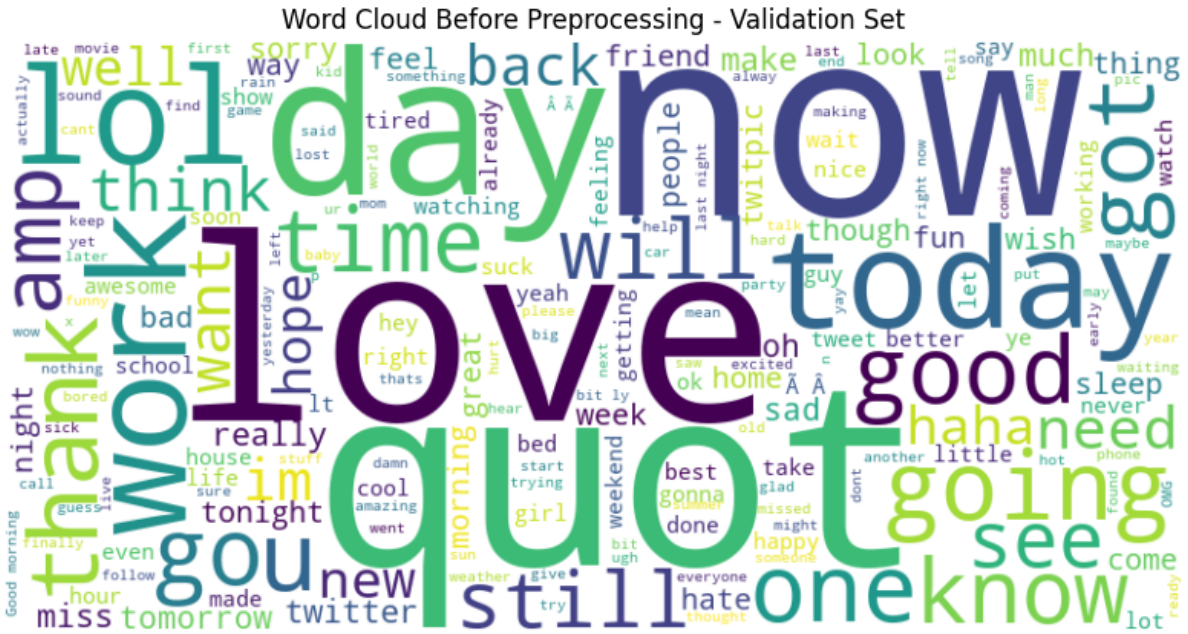

Figure 7: Word Cloud Before Preprocessing (Validation Set)

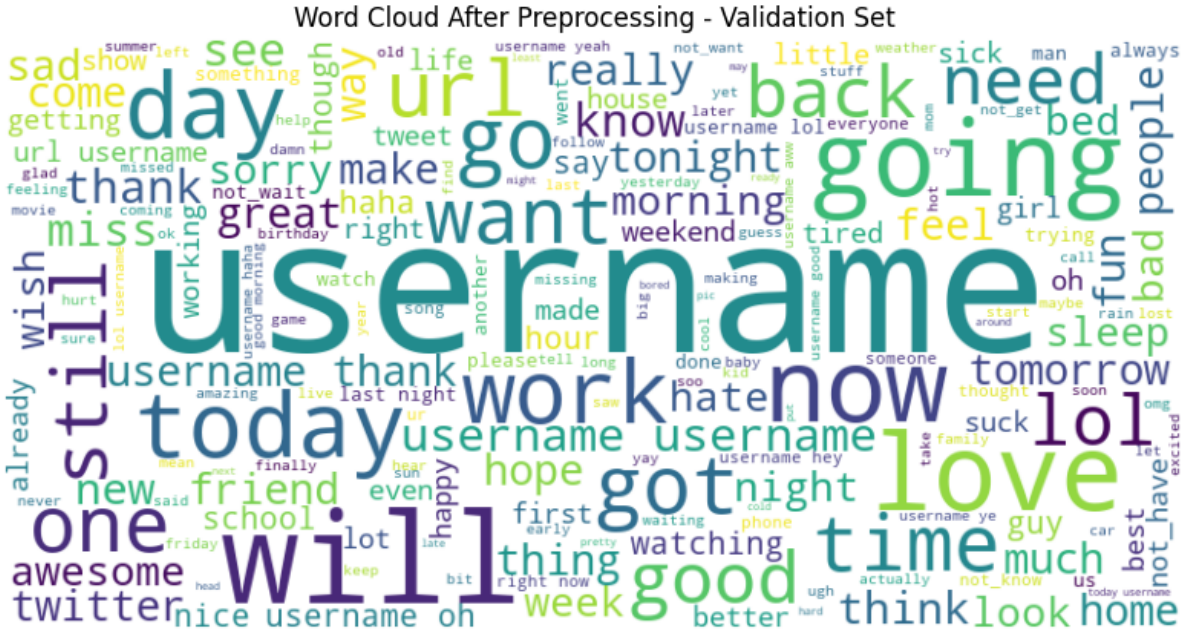

Figure 8: Word Cloud After Preprocessing (Validation Set)

The similarity in word distributions between the training and validation sets ensures that the model is trained and evaluated on **linguistically consistent data**. This consistency reduces the likelihood of unexpected shifts in model performance during validation.

2.2.4. Bigram Analysis. To evaluate the impact of preprocessing on the dataset, frequent bigrams were extracted **before and after preprocessing**. This analysis helps in understanding how text cleaning and transformation affect the most common word pairs.

Key Observations Before Preprocessing:

- **Stop Words Were Overrepresented:** Many of the most frequent bigrams consisted of common stop words, such as `"in the"` (4,200 occurrences), `"for the"` (3,381 occurrences), and `"on the"` (2,992 occurrences). These phrases did not contribute to sentiment and were candidates for removal.
- **Presence of URLs and Non-Linguistic Artifacts:** The bigram `"twitpic com"` appeared frequently (1,852 occurrences), indicating that URLs had a significant impact on word frequency.

Key Changes After Preprocessing:

- **Sentiment-Related Bigrams Became More Prominent:** Frequent bigrams after preprocessing included `"it is"` (11,496), `"do not"` (6,485), `"can not"` (5,967), `"want to"` (2,776), and `"did not"` (2,737), which are more relevant for sentiment classification.
- **Contractions Were Properly Expanded:** The presence of `"do not"`, `"you are"`, `"we will"`, `"it is"`, and `"that is"` confirms that contractions were successfully expanded.
- **Improved Syntactic Structure:** Replacing mentions, expanding contractions, and handling negations allowed better representation of sentiment-focused expressions.

Bigram (Before)	Frequency (Before)	Bigram (After)	Frequency (After)
in the	4200	it is	11496
going to	3944	do not	6485
for the	3381	can not	5967
on the	2992	going to	5809
to be	2846	we will	4544
to go	2803	in the	4119
to the	2761	that is	3432
have to	2676	to be	3402
of the	2320	for the	3367
want to	2139	to go	3313
to get	2130	you are	3113
at the	1980	on the	2959
it was	1868	want to	2776
go to	1862	to the	2753
twitpic com	1852	did not	2737

Table 4: Top 15 Bigrams Before and After Preprocessing (Training Set)

2.2.5. Trigram Analysis. To further assess the impact of preprocessing, frequent trigrams were extracted **before and after preprocessing**. This analysis provides insight into how word sequences evolved after text cleaning.

Key Observations Before Preprocessing:

- **Presence of URLs and Non-Textual Artifacts:** Trigrams such as `"http twitpic com"` (1,833 occurrences) and `"http bit ly"` (1,149 occurrences) were among the most frequent, indicating that URLs heavily influenced word sequence patterns.
- **Informal Expressions and Incomplete Phrases:** Phrases such as `"looking forward to"` (499 occurrences) and `"can wait to"` (381 occurrences) suggested the presence of expressions that could affect sentiment detection.
- **Stop Word Trigrams Were Common:** Sequences like `"to go to"` (914 occurrences) and `"have to go"` (307 occurrences) were frequent.

Key Changes After Preprocessing:

- **URLs Were Successfully Removed:** URL-related trigrams, such as `"http twitpic com"` and `"http bit ly"`, were eliminated because of the replacement with "url".
- **Sentiment-Related Trigrams Became More Frequent:** The most common trigrams after preprocessing included `"am going to"` (1,493 occurrences), `"can not not_wait"` (1,291), and `"is going to"` (1,036), all of which reflect expressions of sentiment or intent.
- **User Mentions Were Replaced:** The token `"username"` appears in trigrams such as `"username username username"` (1,057 occurrences) and `"username thank you"` (649), preserving sentence structure while anonymizing user references.
- **Negation Handling Was Effective:** Trigrams such as `"do not not_know"` (957) and `"not not_want to"` (863) reflect explicit negation propagation, ensuring better sentiment understanding.

Trigram (Before)	Frequency (Before)	Trigram (After)	Frequency (After)
http twitpic com	1833	am going to	1493
http bit ly	1149	can not not_wait	1291
to go to	914	to go to	1073
http tinyurl com	642	username username username	1057
thanks for the	601	is going to	1036
http plurk com	578	username it is	975
going to be	567	going to be	964
looking forward to	499	do not not_know	957
is going to	455	not not_want to	863
go to the	385	username that is	860
can wait to	381	do not not_want	851
want to go	377	it is not	798
how are you	312	username do not	742
have to go	307	username we will	710
going to bed	306	thanks for the	649

Table 5: Top 15 Trigrams Before and After Preprocessing (Training Set)

Validation Set Consistency: A similar pattern was observed in the validation set, where frequent bigrams and trigrams before preprocessing were dominated by stop words, URLs, and generic phrases. After preprocessing, more sentiment-relevant expressions emerged while repetitive `"username"` tokens, remained. This consistency between training and validation sets confirms that preprocessing steps were applied uniformly.

Bigram (Before)	Frequency (Before)	Bigram (After)	Frequency (After)
in the	1249	it is	3316
going to	1127	do not	1820
for the	938	can not	1760
on the	867	going to	1650
to be	818	we will	1226
to the	794	in the	1221
to go	757	to be	979
have to	720	that is	974
of the	655	for the	935
it was	600	you are	899
to get	598	to go	896
at the	579	on the	856
on my	555	we are	806
want to	533	to the	794
to see	529	is not	776

Table 6: Top 15 Bigrams Before and After Preprocessing (Validation Set)

Trigram (Before)	Frequency (Before)	Trigram (After)	Frequency (After)
http twitpic com	501	am going to	417
http bit ly	315	can not not_wait	375
to go to	238	username username username	305
thanks for the	186	is going to	302
http tinyurl com	165	to go to	286
going to be	155	username it is	282
http plurk com	142	going to be	280
looking forward to	134	username that is	271
is going to	131	do not not_know	248
can wait to	121	not not_want to	225
want to go	104	do not not_want	225
how are you	103	it is not	220
in the morning	99	username do not	211
go to the	93	do not not_have	211
going to the	88	username we will	201

Table 7: Top 15 Trigrams Before and After Preprocessing (Validation Set)

2.2.6. Alphanumeric Word Analysis. Key Observations Before Preprocessing:

- Frequent slang expressions such as `"2day"` (today), `"gr8"` (great), `"b4"` (before), and `"h8"` (hate) appeared in the dataset.

- Many of these terms represent strong sentiment (e.g., `"h8"` → `"hate"`), meaning that replacing them correctly can impact sentiment classification.
- Some abbreviations like `"w/o"` (without) and `"w/"` (with) could also appear and affect grammatical understanding if left unchanged.

Preprocessing Decisions

- **Slang Words Mapped to Standard English:** Informal words were expanded to their full forms, ensuring consistency in sentiment-bearing words.
- **Retention of Certain Abbreviations:** Some widely recognized abbreviations (e.g., `"lol"`, `"omg"`) were retained because replacing them led to a decrease in model accuracy.
- **Handling of Alphanumeric Abbreviations:** Common numeric abbreviations (e.g., `"2nite"` → `"tonight"`, `"1st"` → `"first"`) were converted into standard words.

Impact on Model Performance

- **Improved Feature Representation:** Expanding contractions allowed the model to capture more meaningful patterns, particularly in sentiment-related phrases.
- **Maintaining Sentence Structure:** Unlike simple removal, replacing slang words preserved sentence coherence, preventing loss of contextual sentiment.
- **Avoiding Unnecessary Expansions:** Over-expanding certain expressions (e.g., replacing "lol" with "laughing out loud") resulted in decreased accuracy, so such terms were retained in their original form.

By normalizing slang words appropriately, the preprocessing step ensured better input consistency while preserving sentiment-indicating features in the dataset.

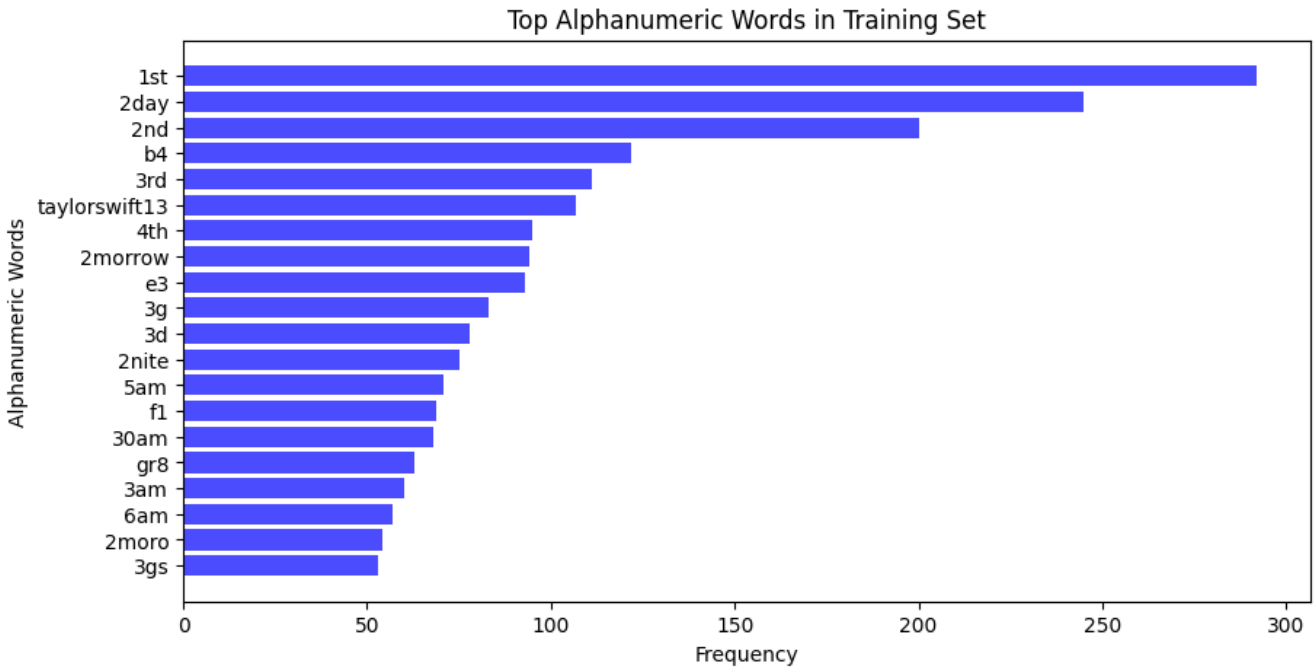

Figure 9: Top Alphanumeric Words in Training Set

Validation Set Consistency The validation set exhibited a similar distribution of alphanumeric words, reinforcing the consistency of such patterns across datasets. The most common terms in the validation set also included ordinal numbers (`"1st"`, `"2nd"`), time-related expressions (`"30am"`, `"3am"`), and abbreviations (`"2day"`, `"2morrow"`).

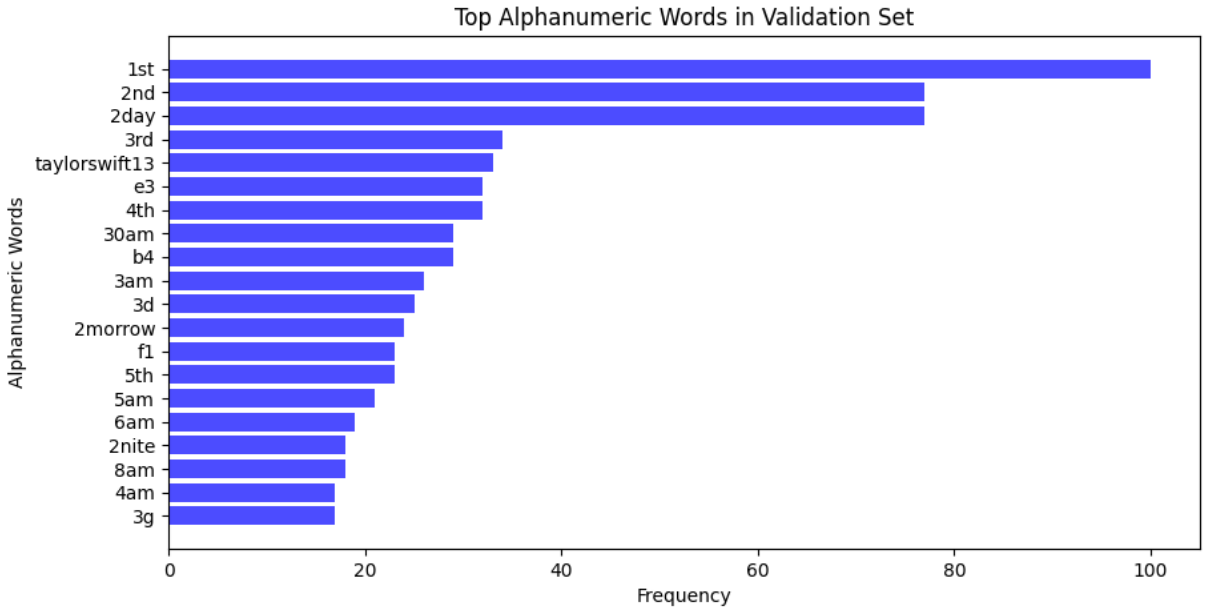

Figure 10: Top Alphanumeric Words in Validation Set

2.2.7. HTML Entities. **Key Observations Before Preprocessing:** HTML entities were frequently found in the dataset, indicating that some text data was not properly decoded. The most common entities observed were:

- `&quot;` (escaped quotation mark) – 6,801 occurrences
- `&` (escaped ampersand) – 4,559 occurrences
- `<` (escaped less-than symbol) – 1,759 occurrences
- `>` (escaped greater-than symbol) – 669 occurrences

These entities can disrupt text processing, making it harder for the model to correctly interpret relationships between words. If left unprocessed, they may interfere with tokenization and feature extraction.

Preprocessing Decisions:

- **Decoding HTML Entities:** All occurrences of encoded symbols were replaced with their respective characters or meaning. For example, `&` was converted to ‘"and"‘ and `>` to ‘">"‘.
- **Special Handling of `>`:** It was noticed that a significant percentage of `>` instances were combined with ‘3‘, representing the heart symbol (‘<3‘), which was replaced with the keyword "love".

2.3. Vectorization

In order to apply Logistic Regression for sentiment classification, the textual data needed to be transformed into a numerical format. For this purpose, the **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorizer was used.

TF-IDF is an effective method for feature extraction from text, as it captures not just how frequently a word appears in a document (term frequency), but also how important that word is across all documents (inverse document frequency).

- **Term Frequency (TF):** Measures how often a term appears in a document.
- **Inverse Document Frequency (IDF):** Penalizes terms that appear too frequently across all documents, thus reducing the weight of non-informative words.

This weighting scheme helps emphasize important, sentiment-related words while reducing the influence of commonly used non-informative words.

3. Algorithms and Experiments

Experimentation Strategy. Following the baseline, a series of incremental experiments were performed to evaluate the impact of various components on model performance. These included:

- **Advanced Preprocessing:** Including slang normalization, emoticon and HTML entity replacement, contraction expansion, and removal of non-informative tokens.
- **TF-IDF Tuning:** Adjusting the n-gram range (e.g., including bigrams), capping the maximum number of features.
- **Hyperparameter Optimization:** Testing different values of regularization strength `C` and other parameters.

Each modification was tested in isolation to evaluate its individual impact using accuracy, precision, recall, and F1-score. The best configuration was then determined by averaging these metrics across validation runs.

3.1. Experiments - Preprocessing

Baseline Model. The first experiment involved training a model using the **raw, unprocessed text data** directly from the training set. No cleaning, normalization, or token refinement was applied. This served as a brute-force attempt to establish a minimal baseline.

- **Vectorizer:** `TfidfVectorizer()`
- **Classifier:** `LogisticRegression(random_state=42)`

- **Preprocessing:** *None*

Results:

Metric	Score
Accuracy	0.7911
Precision	0.7912
Recall	0.7911
F1 Score	0.7911
Average	0.7911

Table 8: Results before preprocessing - Baseline Model

Test Accuracy: 0.78928

These results confirmed that a logistic regression model with TF-IDF can extract useful features even without preprocessing, but also highlighted the limitations of relying on raw text, motivating further improvements.

Experiment: Baseline + Lowercasing. **Objective:** To examine the effect of minimal preprocessing, the baseline model was retrained using the same setup but with lowercase conversion applied to all text. This small change aimed to reduce vocabulary sparsity caused by case variations.

Results:

Metric	Score
Accuracy	0.7911
Precision	0.7912
Recall	0.7911
F1 Score	0.7911
Average	0.7911

Table 9: Results before preprocessing - Baseline Model

Test Accuracy: 0.78928

Note: The `TfidfVectorizer` in `scikit-learn` applies lowercasing by default (`lowercase=True`). As a result, manually lowercasing the input text had no effect on the model's performance. This explains why the metrics for the "Baseline + Lowercasing" experiment were identical to the original baseline.

Experiment: URL Replacement + Lowercasing. **Objective:** This experiment aimed to evaluate whether replacing URLs with the generic token `"url"` after lowercasing text would improve sentiment classification performance.

Results:

Metric	Score
Accuracy	0.7912
Precision	0.7912
Recall	0.7912
F1 Score	0.7912
Average	0.7912

Table 10: Results after lowercasing

Test Accuracy: 0.78881
Analysis:

- The replacement of URLs with a generic token did not lead to an improvement in classification performance.
- This suggests that URLs do not carry significant sentiment-related information and their presence or replacement does not strongly impact the model's ability to differentiate between positive and negative sentiments.
- Removing URLs completely or extracting key parts of URLs (e.g., domain names) were tested, but lowered accuracy

Experiment: Replacing Mentions with "username". **Objective:** This experiment tested whether replacing mentions (e.g., `"@user123"`) with the generic token `"username"` improves sentiment classification. Mentions might carry implicit sentiment, so preserving them as a placeholder rather than removing them entirely was expected to help retain context.
Results:

Metric	Score
Accuracy	0.7912
Precision	0.7913
Recall	0.7912
F1 Score	0.7912
Average	0.7912

Table 11: Results after replacing mentions with "username"

Test Accuracy: 0.78786
Analysis:

- Removing mentions entirely causes information loss, while replacing them with `"username"` allows the model to retain structural and contextual clues.
- The small but measurable improvement suggests that mentions should be kept in a neutral form rather than being discarded.

Experiment: Repetition Normalization and "xx" Replacement. **Objective:** This experiment involved two preprocessing steps:

- Reducing repeated characters to a maximum of two (e.g., `"soooo"` → `"soo"`).

- Replacing occurrences of `"xx"` with the token `"kiss"`, based on its frequent use to express affection.

Results:

Metric	Score
Accuracy	0.7917
Precision	0.7918
Recall	0.7917
F1 Score	0.7917
Average	0.7917

Table 12: Results after reducing character repetition and replacing "xx" with "kiss"

Test Accuracy: 0.78895
Analysis:

- Normalizing exaggerated expressions (e.g., `"sooo happy"`) improves consistency in word representation without altering their sentiment.
- Interpreting `"xx"` as affectionate and mapping it to `"kiss"` preserves valuable sentiment-related features, leading to better feature recognition during classification.
- These transformations offered a measurable gain in performance.

Experiment: Decoding, Emoticon Replacement, and Punctuation Handling. **Objective:** This experiment combined several preprocessing techniques:

- Applying `ftfy.fix_text()` to fix non-ASCII artifacts and encoding issues.
- Replacing certain emoticons (e.g., `"<3"` → `"love"`) with sentiment-bearing keywords.
- Removing `',@,*` and replacing other punctuation marks with whitespace to ensure cleaner tokenization.

Results:

Metric	Score
Accuracy	0.7934
Precision	0.7934
Recall	0.7934
F1 Score	0.7934
Average	0.7934

Table 13: Results after Decoding, Emoticon Replacement, and Punctuation Handling

Test Accuracy: 0.78980
Analysis:

- Replacing emoticons and fixing non-ASCII artifacts helped reduce noise and enhanced semantic clarity for the model.
- Removing ',@,* and normalizing punctuation improved token consistency.

Experiment: Slang Normalization, Percentage Mapping, and Numeric Cleanup. **Objective:** This experiment focused on enriching text normalization by replacing slang and abbreviated words with their standard forms, mapping common percentage expressions to meaningful keywords, and removing standalone numbers and one-character words that contributed little to sentiment understanding.

Preprocessing Steps:

- **Slang and Abbreviation Expansion:** Slang words like `"luv"`, `"fkin"`, `"wat"`, and abbreviations like `"plz"`, `"ty"`, `"frnd"` were mapped to their full forms (e.g., `"luv"` → `"love"`).
- **Percentage Mapping:** Phrases such as `"50%"`, `"25%"`, and `"100%"` were mapped to interpretable tokens like `"half"`, `"quarter"`, and `"all"`.
- **Noise Reduction:** All remaining numeric tokens and one-character words (e.g., `"1"`, `"a"`, `"x"`) were removed to reduce noise and irrelevant features.

Results:

Metric	Score
Accuracy	0.7938
Precision	0.7939
Recall	0.7938
F1 Score	0.7938
Average	0.7938

Table 14: Results after slang normalization, percentage mapping, and numeric cleanup

Test Accuracy: 0.79060
Analysis:

- The added slang coverage improved semantic clarity of the input, allowing the model to better capture sentiment.
- Mapping numerical percentages into words (e.g., `"half"`, `"all"`) gave the model more interpretable features.
- Removing irrelevant characters and numbers helped reduce feature sparsity, contributing to the observed improvement in accuracy and F1 score.

Experiment: Contraction Expansion and Negation Handling. **Objective:** This experiment aimed to enhance sentiment detection by expanding informal contractions (e.g., `"don't"` → `"do not"`) and introducing explicit negation tagging. Since negations often reverse sentiment, tracking their scope was expected to improve classification.

Preprocessing Details:

- **Contraction Expansion:** Common contractions were expanded into their complete forms using a custom dictionary (e.g., `"can't"` → `"cannot"`, `"i'm"` → `"i am"`).
- **Negation Handling:** After a negation word (e.g., `"not"`, `"never"`, `"without"`), the next relevant word was modified with a prefix `"not_"` (e.g., `"not happy"`, `"without help"` → `"not_help"`), skipping neutral modifiers like `"too"`, `"very"`.

Results:

Metric	Score
Accuracy	0.7995
Precision	0.7996
Recall	0.7995
F1 Score	0.7995
Average	0.7995

Table 15: Results after contraction expansion and negation handling

Test Accuracy: 0.79706
Analysis:

- Negation awareness allowed the model to distinguish between positive and negated contexts (e.g., `"not good"`).
- Expanding contractions ensured consistent vocabulary and helped TF-IDF capture more meaningful features.
- This combination resulted in the best performance so far, indicating that grammatical clarity and negation sensitivity are crucial for sentiment analysis.

Failure Cases and Observations.

- **Stopword Removal:** Initially removing all stopwords decreased accuracy. Certain stopwords contributed sentiment (e.g., "not," "never"), so selective retention was applied.
- **Replacing Hashtags/Names with Tokens:** Replacing all hashtags or names with "hashtag" or "name" hurt performance. These words sometimes held sentiment cues (e.g., #love), so this step was reverted.
- **Excessive Cleaning:** Removing too many special tokens or applying aggressive stemming reduced model expressiveness.

Visualizations - Preprocessing

Learning Curves: Before and After Preprocessing. The following plots illustrate the learning behavior of the baseline model trained on raw vs. preprocessed text. These visualizations help quantify the benefit of preprocessing on model generalization.

Before Preprocessing:

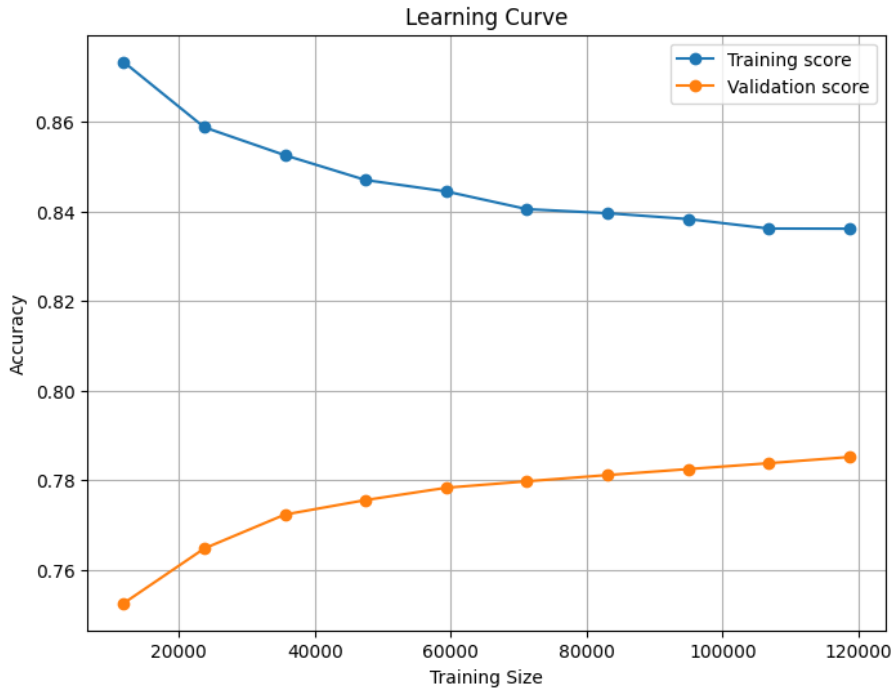

Figure 11: Learning Curve of Baseline Model (Before Preprocessing)

The baseline model trained on unprocessed text shows signs of overfitting. The training accuracy is high, but validation accuracy remains relatively low and flat, suggesting the model fails to generalize well from noisy input data.

After Preprocessing:

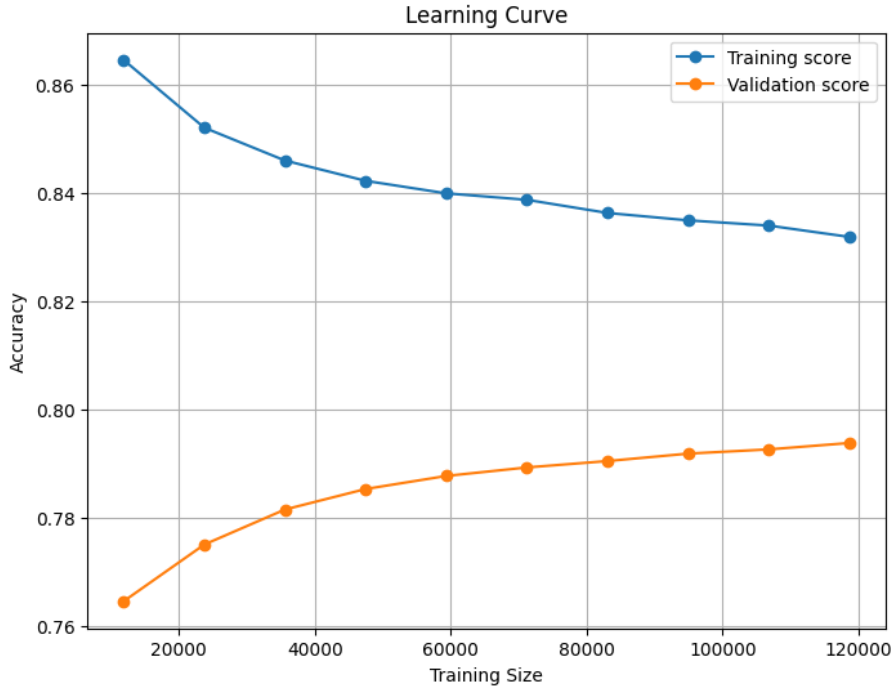

Figure 12: Learning Curve After Preprocessing (Default TF-IDF + Logistic Regression)

Preprocessing leads to improved validation accuracy and a more stable learning

curve. The gap between training and validation accuracy is narrower, indicating better generalization.

Key Insights:

- Preprocessing consistently improved validation performance.
- The cleaned data allowed the model to learn more generalizable patterns.

Confusion Matrices: Before and After Preprocessing. In addition to learning curves, confusion matrices were examined to assess the model's classification performance in more detail.

Before Preprocessing:

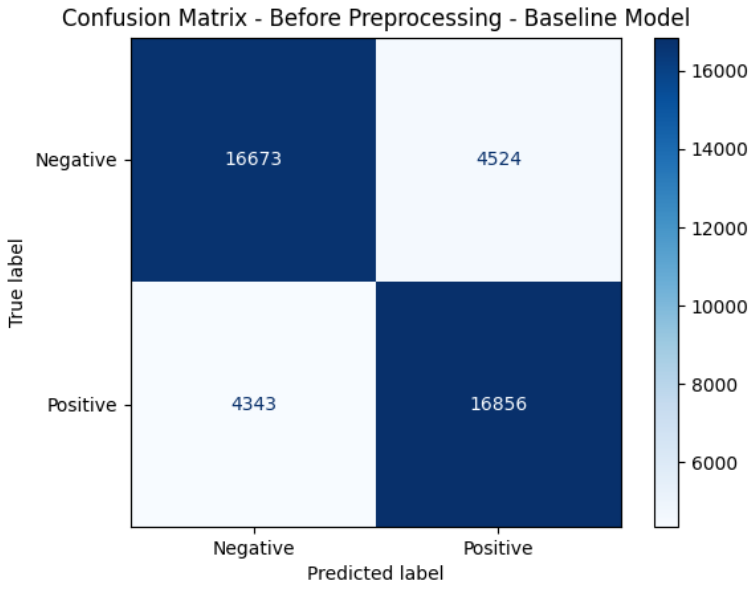

Figure 13: Confusion Matrix – Baseline Model Before Preprocessing

The baseline model misclassified a considerable number of both positive and negative examples. The relatively high number of false positives and false negatives indicates that the model had difficulty capturing the correct decision boundary due to the noisy input data.

After Preprocessing:

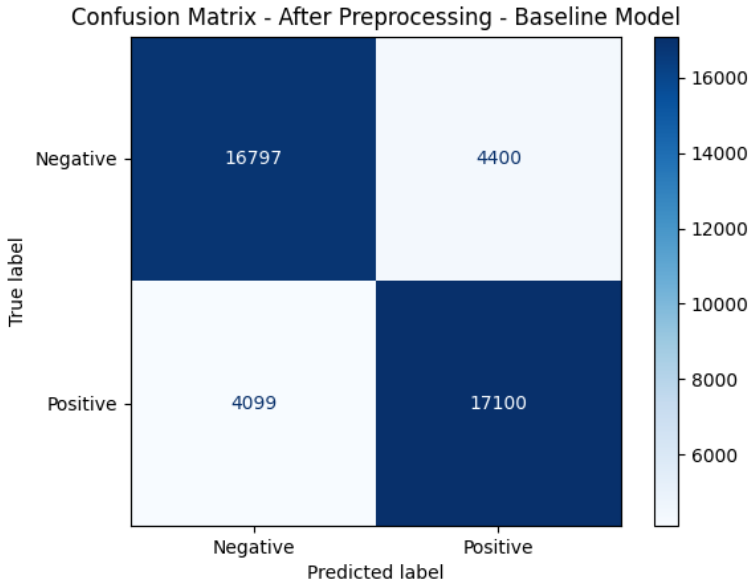

Figure 14: Confusion Matrix – Baseline Model After Preprocessing

After preprocessing, the model correctly classified a higher number of samples in both classes. The reduced number of misclassifications highlights that the model learned more effective patterns from the cleaned text data.

Key Insights:

- Preprocessing helped the model reduce both false positives and false negatives.
- Classification became more balanced, with improved sensitivity and specificity for both sentiment classes.
- Overall, the confusion matrix confirms the quantitative improvements observed in accuracy, precision, recall, and F1-score.

ROC Curve: Before and After Preprocessing. The ROC curves below visualize the trade-off between true positive and false positive rates.

ROC Curve Before Preprocessing:

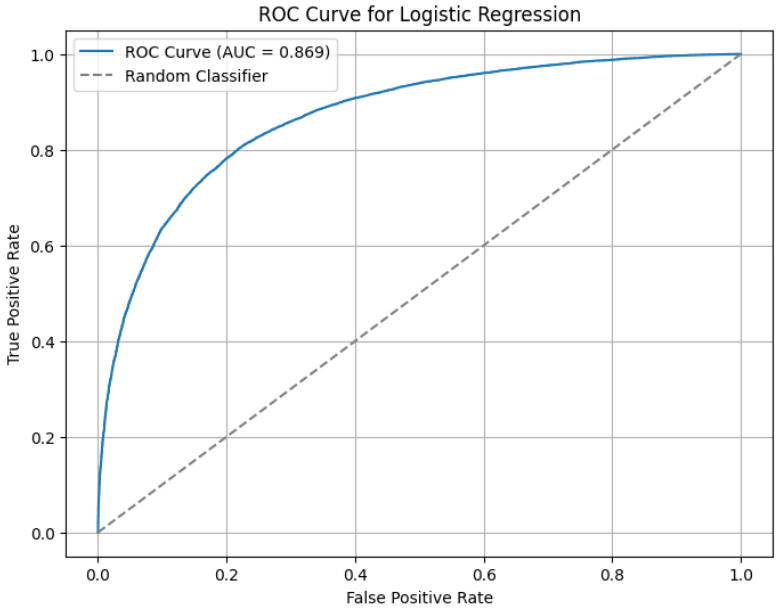

Figure 15: ROC Curve - Baseline Model (Before Preprocessing), AUC = 0.869

ROC Curve After Preprocessing:

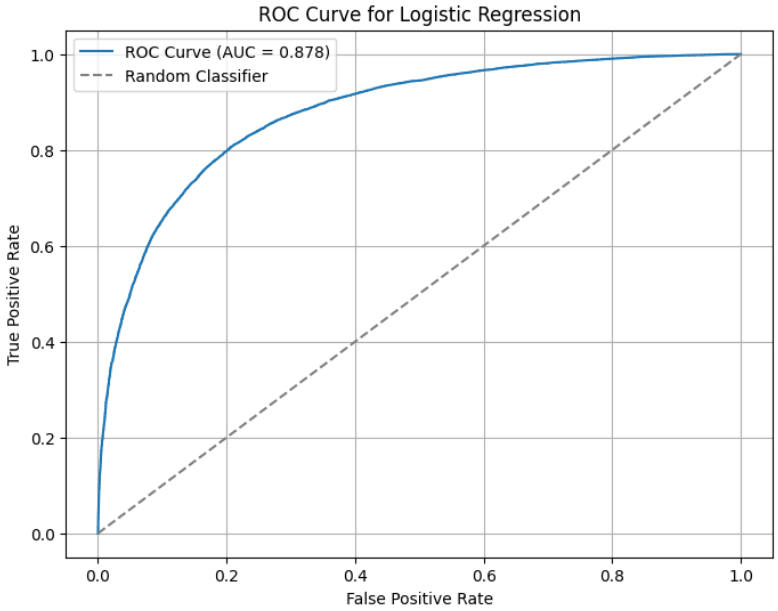

Figure 16: ROC Curve - Baseline Model (After Preprocessing), AUC = 0.878

Conclusion:

- Preprocessing improved the model's Area Under the Curve (AUC), indicating better discrimination between classes.

- This supports the earlier findings from accuracy, F1 score, and confusion matrix evaluations.

3.2. Experiments - Hyper-parameter Tuning

After establishing a strong baseline using fully preprocessed data, hyperparameter optimization was conducted to improve model performance. The goal was to fine-tune both the TF-IDF vectorizer and the Logistic Regression classifier.

Baseline for Tuning. The starting point for tuning was the fully preprocessed model trained with default settings:

- **Vectorizer:** `TfidfVectorizer()` (default parameters)
- **Classifier:** `LogisticRegression()` with `max_iter=1000`
- **Validation Accuracy: 0.7995**

Tuning Strategy. To enhance this baseline, the following hyperparameters were explored:

- **TF-IDF Vectorizer:**
  - `ngram_range` – inclusion of bigrams and trigrams
  - `max_features` – limiting vocabulary size to the most informative terms
  - `min_df`, `max_df` – filtering rare or overly common tokens
  - `sublinear_tf` – logarithmic term frequency scaling
  - `norm` – normalization method (`"l1"` or `"l2"`)
- **Logistic Regression:**
  - `C` – inverse of regularization strength
  - `penalty` – regularization type (`"l1"`, `"l2"`)
  - `solver` – optimization algorithm

Hyperparameter search was performed using the `Optuna` framework, optimizing for the weighted F1-score via 5-fold cross-validation.

Study 1: Broad Search. The first study covered a wide range of values to identify promising regions of the search space. Parameters included:

- `max_features`: [55,000 – 70,000]
- `ngram_upper`: [1 – 3]
- `min_df`: [1 – 10]
- `max_df`: [0.85 – 1.0]

- `C`: [0.5 – 3.0] (log scale)
- `solver`: [`"liblinear"`, `"lbfgs"`]

The best configuration from this study achieved a weighted F1-score of **0.8048**:

- `max_features = 65384`
- `ngram_upper = 3`
- `min_df = 2`
- `max_df = 0.9415`
- `C = 1.3006`
- `solver`: [`"liblinear"`

Study 2: Refined Search. The second study focused on a narrower range based on prior observations, enabling sublinear term frequency and exploring solver options.

- `max_features`: [61,000 – 68,000]
- `ngram_upper`: [2 – 3]
- `min_df`: [1 – 3]
- `max_df`: [0.90 – 1.0]
- `C`: [1.0 – 2.0]
- `solver`: [`"liblinear"`, `"lbfgs"`]
- Fixed: `sublinear_tf = True`, `norm = "l2"`, `penalty = "l2"`

This refined search resulted in the best overall weighted F1-score of **0.8054**:

- `max_features = 67000`
- `ngram_upper = 3`
- `min_df = 1`
- `max_df = 0.97`
- `sublinear_tf = True`
- `norm = "l2"`
- `C = 1.3579`
- `penalty = "l2"`
- `solver = "liblinear"`

3.3. Final Evaluation

To further analyze the performance of the final model, both a confusion matrix and a learning curve were generated.

Confusion Matrix. The confusion matrix illustrates the distribution of predictions over actual labels on the test set. The model performs well, with strong diagonal dominance indicating correct predictions for both classes.

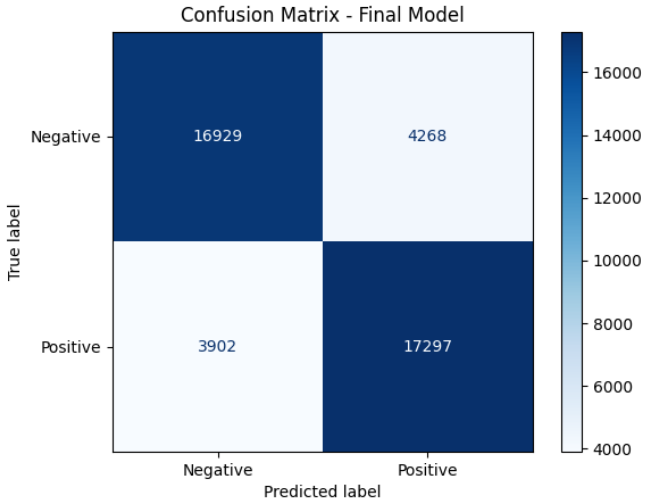

Figure 17: Confusion Matrix for the Final Preprocessed Model

Learning Curve. The learning curve below shows model accuracy as a function of training set size. The validation curve stabilizes and narrows the gap with the training curve, suggesting the model benefits from additional data and is not overfitting.

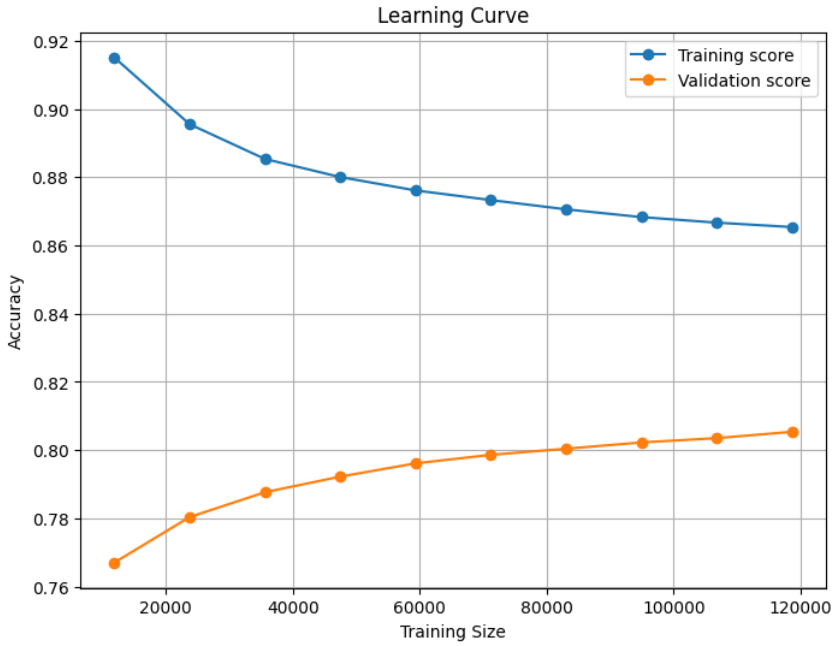

Figure 18: Learning Curve for Final Model (TF-IDF + Logistic Regression)

ROC Curve. The ROC curve evaluates the model's performance across different classification thresholds. The Area Under the Curve (AUC) is **0.889**, indicating strong discriminatory capability.

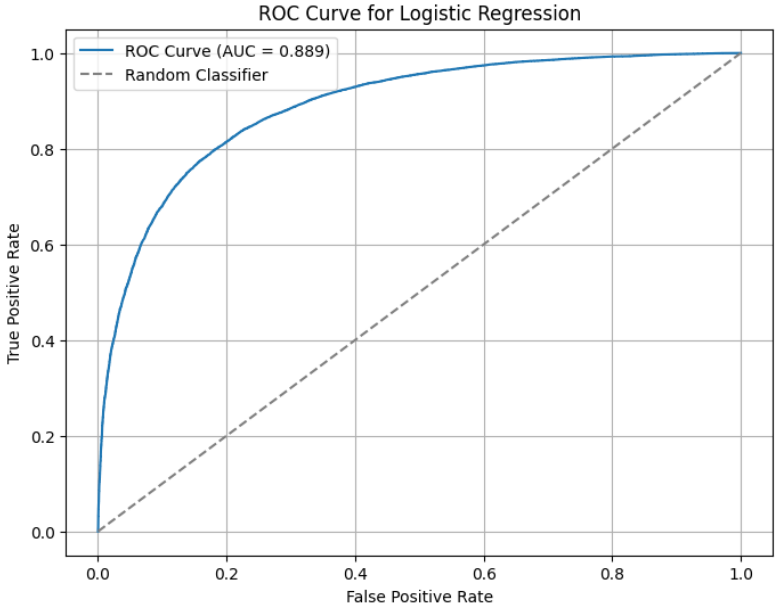

Figure 19: ROC Curve for Final Model (AUC = 0.889)

3.4. Conclusion

The final model, configured using the best hyperparameters from the refined Optuna study, achieved a test accuracy of **0.80947**. This result surpasses the original baseline,

confirming that systematic preprocessing and targeted hyperparameter tuning significantly improved performance. The combination of TF-IDF feature engineering and Logistic Regression, optimized via cross-validated F1-score, yielded a well-generalizing and interpretable sentiment classifier.