UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Andreakis Dimitrios*
sdi2300008

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

Contents

1. Abstract

Sentiment analysis is a crucial task in Natural Language Processing (NLP), enabling the classification of text into positive or negative sentiments. In this project, we aim to develop a sentiment classification model using logistic regression. The dataset consists of labeled tweets, which are preprocessed and transformed into numerical representations using TF-IDF vectorization. We employ exploratory data analysis (EDA) to understand the dataset's structure and characteristics.

To optimize model performance, we experiment with different hyperparameters, evaluate results using multiple metrics (accuracy, precision, recall, and F1-score), and analyze learning curves. The final model is fine-tuned to maximize accuracy, ensuring robustness across different data splits. The results are assessed comprehensively to provide insights into the model's strengths and limitations.

2. Data processing and analysis

2.1. Pre-processing

To prepare the dataset for sentiment classification, we applied several pre-processing steps to clean and standardize the text data.

The following techniques were used:

- Lowercasing: All text was converted to lowercase to ensure uniformity and prevent duplicate representations of the same word (e.g., "Good" and "good").

- Removing URLs: Links and URLs were removed as they do not contribute meaningful semantic information for sentiment classification.

- Removing special characters and punctuation: Non-alphanumeric symbols, such as emojis and punctuation marks, were eliminated to simplify tokenization.

- Tokenization: Sentences were split into individual words to allow efficient vectorization.

- Stopword removal: Commonly used words (e.g., "the", "is", "and") that do not carry significant sentiment information were removed.

- Lemmatization: Words were reduced to their base forms (e.g., "running" $\rightarrow$ "run") to improve model generalization.

However, we kept only "Lowercasing", as the other techniques seemed to lower the accuracy of the model.

After pre-processing, the cleaned text data was transformed into numerical representations using TF-IDF vectorization, which enabled effective feature extraction for model training.

2.2. Analysis

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

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

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

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

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

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

A visualization of the class distribution is provided below:

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

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

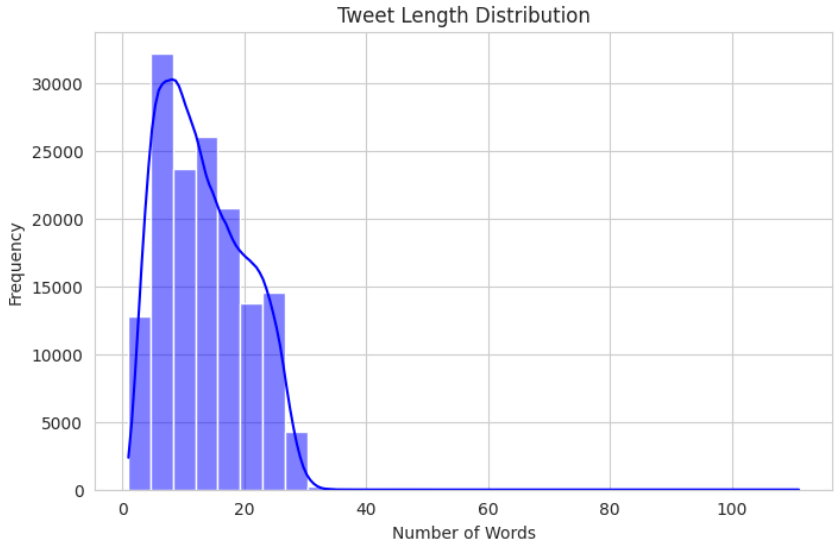

Figure 2: Histogram of tweet length distribution.

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

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

Figure 3: Word cloud of positive sentiment tweets.

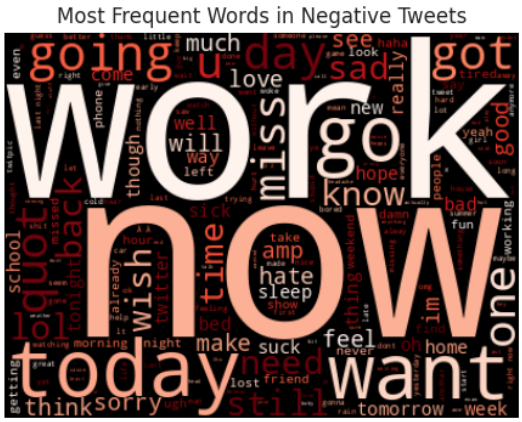

Figure 4: Word cloud of negative sentiment tweets.

2.2.5. Key Observations.

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

2.3. Data partitioning for train, test and validation

The dataset was provided in three pre-defined partitions:

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

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

2.4. Vectorization

For this task, we used the **Term Frequency-Inverse Document Frequency (TF-IDF)** technique to transform the text data into numerical features suitable for machine learning models.

2.4.1. What is TF-IDF?. TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a corpus of documents. It emphasizes words that are frequent in a document but rare in the overall dataset, helping to capture important and unique terms.

2.4.2. How TF-IDF Works. TF-IDF consists of two main components:

- **Term Frequency (TF):** Measures how often a word appears in a document.

$$\text{TF}(t,d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in d}}$$

- **Inverse Document Frequency (IDF):** Measures how important a word is across all documents.

$$\text{IDF}(t) = \log\left(\frac{N}{DF(t)}\right)$$

  where N is the total number of documents, and $DF(t)$ is the number of documents containing term t. The log function is used to prevent overly frequent terms from dominating the importance score.

The formula for TF-IDF is:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

2.4.3. Why Use Logarithm in IDF?. The use of a logarithm ensures that very common words (like "the", "is") do not overpower the importance of rarer terms. Without logarithmic scaling, the difference in IDF scores would be much more exaggerated, which could lead to an unstable representation of word importance.

2.4.4. Drawbacks of TF-IDF. Although TF-IDF is effective in many cases, it has some limitations:

- It doesn't capture meaning or context—synonyms like "happy" and "joy" are treated as distinct words.
- It doesn't account for word order or phrases—e.g., "machine learning" is treated as two separate words.
- It may give high importance to very rare words, even if they are irrelevant or misspelled.
- It doesn't distinguish between different contexts of the same word (e.g., "Apple" the fruit vs. "Apple" the company).

3. Algorithms and Experiments

3.1. Experiments

To tackle the sentiment analysis problem, we started by testing different pre-processing techniques and vectorization methods. We followed a structured approach where we first applied basic pre-processing techniques and evaluated their impact on the accuracy of our model. Subsequently, we experimented with various configurations of the `TfidfVectorizer` and `LogisticRegression` parameters to see if further optimizations could improve the model's performance.

3.1.1. Pre-Processing Techniques. We began by testing multiple preprocessing techniques to see how they influenced the model's accuracy. The preprocessing methods include:

- Lowercase all text
- Remove URLs, HTML tags, special characters, and punctuation
- Tokenization
- Remove stop words
- Lemmatization and SymSpell (spell correction)

We tried the following combinations of these preprocessing techniques and evaluated the accuracy:

- **Current Full Preprocessing:** Full preprocessing with spell correction, lemmatization, punctuation removal, and stop word removal.

- **Minimal Cleaning:** Only punctuation removal and stop word removal (no lemmatization or spell correction).
- **Without Stopword Removal:** Kept stop words to see if they help in sentiment/-context analysis.
- **Without Lemmatization:** Kept original words without applying lemmatization.
- **Without Spell Correction:** Omitted SymSpell to see if over-correction affects performance.
- **Lowercasing + Tokenization Only:** Only lowercase and tokenization, with minimal cleaning.

The accuracy results for these preprocessing variations were as follows:

- **Current Full Preprocessing:** 77.5474%
- **Minimal Cleaning:** 77.4248%
- **Without Stopword Removal:** 77.2903%
- **Without Lemmatization:** 77.2667%
- **Without Spell Correction:** 77.2809%
- **Lowercasing + Tokenization Only:** 77.2809%
- **Used Stemming:** 77.9295%
- **Used Both Lemmatization and Stemming (with some stopwords):** 77.8965%
- **Used Stemming and Kept Some Stop Words:** 77.9295%
- **Only Lowercasing:** 79.5759%

3.1.2. TfidfVectorizer Experiments. Next, we focused on the vectorization step. Initially, we tested the `TfidfVectorizer` without any additional parameters and observed the performance:

- **0 Parameters (Basic TF-IDF):** 79.1136%

We then experimented with different n-gram ranges, maximum document frequency (`max_df`), and minimum document frequency (`min_df`) parameters to improve the results. These configurations yielded the following results:

- **Ngram (1-3):** 79.5759%
- **max_df = 0.95, min_df = 2:** 80.2552%
- **max_df = 0.95, min_df = 3:** 80.2646%
- **Sublinear TF Scaling:** 80.2788%
- **Ngrams (1-2):** 80.2977%

3.1.3. Logistic Regression Optimization. Following the vectorization, we applied Logistic Regression for classification. We tested different hyperparameters for the Logistic Regression model. The results are summarized below:

- **Random State:** 80.2977%
- **C = 1.4 (Regularization Parameter):** 80.4722%
- **SMOTE (Synthetic Minority Over-sampling Technique):** Using `smote.fit_resample` resulted in a reduced accuracy of 80.3849%.

3.2. Hyper-parameter tuning

After experimenting with various preprocessing techniques and vectorization configurations, we focused on fine-tuning the hyperparameters for Logistic regression. Specifically, we adjusted the `C` parameter, which controls the regularization strength in the Logistic regression model. A higher value of `C` results in less regularization, while a lower value increases regularization. We found that setting `C = 1.4` gave the best results, increasing accuracy to 80.4722%. The final model using these settings achieved an accuracy of 80.4722% on the validation set.

We also experimented with the oversampling technique SMOTE, but the results showed a slight decrease in accuracy (80.3849%), indicating that balancing the dataset in this case did not improve model performance.

3.3. Optimization techniques

In terms of optimization, we primarily focused on optimizing the model through parameter tuning. The adjustments made to the `TfidfVectorizer`, such as changing `max_df`, `min_df`, and using sublinear scaling, led to the most notable improvements in accuracy. The vectorizer was the key optimization technique, as it directly impacted the quality of the feature representation from the raw text data.

Additionally, the regularization parameter `C` in the Logistic Regression model played a significant role in improving the results.

3.4. Evaluation

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

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

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

The evaluation metrics are defined as follows:

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

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

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

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

3.4.2. Model Performance Results. Table 1 summarizes the final model's evaluation metrics:

Metric	Score
Accuracy	0.8047
Precision	0.8000
Recall	0.8100
F1 Score	0.8050

Table 1: Final model performance metrics

3.4.3. Classification Report. The detailed classification report for each sentiment class (positive and negative) is presented in Table 2:

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

Class	Precision	Recall	F1-Score	Support
Negative (0)	0.81	0.80	0.80	21197
Positive (1)	0.80	0.81	0.81	21199
Macro Avg	0.80	0.80	0.80	42396
Weighted Avg	0.80	0.80	0.80	42396

Table 2: Classification report of the final model

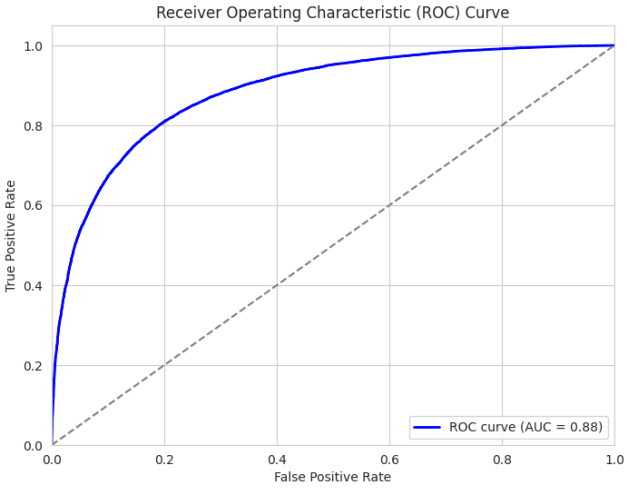

Figure 5: ROC Curve

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

3.4.6. Confusion Matrix. The confusion matrix provides a summary of the prediction results. It highlights the number of correct and incorrect classifications. It is visualized in Figure 7.

Observations:

- The model achieved an **accuracy of 80.47%**, which indicates strong performance.
- The **precision (80%) and recall (81%)** show a balance between false positives and false negatives.
- The **F1-score (80.5%)** confirms this balance, making the model effective in handling sentiment classification.
- Since the dataset has an almost equal number of positive and negative samples, accuracy is a reliable metric in this case.

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

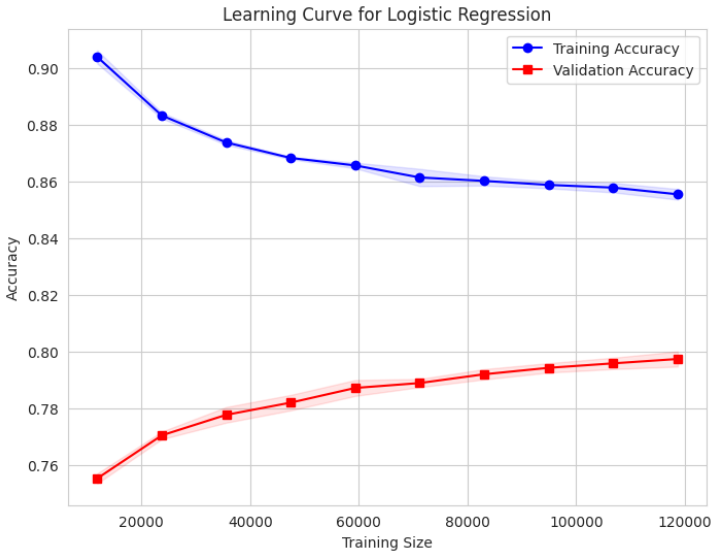

Figure 6: Learning Curve

4. Results and Overall Analysis

4.1. Results Analysis

The model's performance was evaluated using various metrics, including **accuracy**, **precision**, **recall**, **F1 score**, and visualizations such as the **ROC curve** and **confusion matrix**.

4.1.1. Accuracy and Classification Report. The model achieved an **accuracy of 80.47%** on the validation set, which is a promising result. The detailed classification report also shows that the precision, recall, and F1-score are around **0.80** for both classes (positive and negative sentiment).

This indicates that the model is able to distinguish between positive and negative sentiment quite well. However, the relatively balanced performance in terms of precision and recall shows that there might be room for improvement, particularly in terms of capturing more positive/negative instances without significantly increasing false positives or false negatives.

4.1.2. Confusion Matrix. The confusion matrix further supports the results, where we observe:

- **True Positives (TP)** = 17,127, meaning the model correctly predicted 17,127 positive sentiments.
- **True Negatives (TN)** = 16,990, indicating the correct identification of negative sentiments.
- **False Positives (FP)** = 4,207, which signifies that 4,207 instances of negative sentiment were wrongly classified as positive.
- **False Negatives (FN)** = 4,072, which highlights that 4,072 positive sentiment instances were incorrectly predicted as negative.

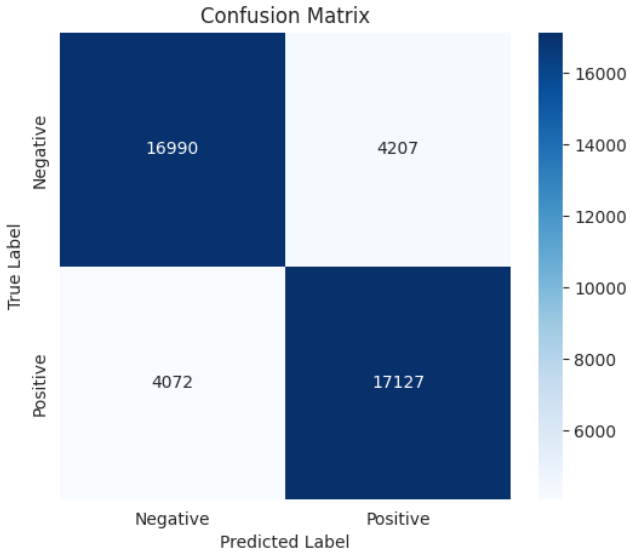

Figure 7: Confusion Matrix

These results suggest that the model's performance is relatively good, but there is still a trade-off between **false positives** and **false negatives**. Improving the model could involve exploring techniques to reduce this error, such as handling class imbalance or tuning the model's hyperparameters.

4.1.3. ROC Curve and AUC. The **ROC curve** for the model demonstrates an **AUC score of 0.88**, which indicates a good performance with a high degree of separability between the two classes (positive and negative). An AUC closer to **1** represents a better performance. In this case, an AUC of **0.88** is a solid result and confirms that the model is performing well, but there is still room for optimization.

4.1.4. Learning Curve. The **learning curve** shows that as the training size increases, the model's accuracy on both training and validation sets stabilizes.

4.1.5. Could More Experiments be Done?. Yes, there are several avenues for improvement and further experiments that could be explored. For example:

- **Neural Networks:** One potential direction for future experimentation would be to explore neural networks, which could potentially capture more complex patterns in the text data.

4.1.6. Conclusion. In conclusion, the model performed reasonably well with an accuracy of 80.47% on the validation set. Although the results are promising, further refinement and experimentation could help improve the model's ability to handle false positives and negatives.

4.1.7. Best trial. The best trial was achieved using the following configuration:

- **Pre-processing:** Lowercasing Only

- **Vectorizer:** TF-IDF with n-grams (1-2), max_df = 0.95, min_df = 3, sublinear_tf = True
- **Model:** Logistic Regression with regularization strength $C = 1.4$
- **Final Accuracy:** 80.472%

This configuration resulted in the highest accuracy achieved during experimentation. The combination of minimal pre-processing and tuned hyperparameters led to a robust model that effectively balanced training and validation performance.

4.2. Comparison with the first project

In this project, we developed a sentiment classifier using deep neural networks for the English-language Twitter dataset. In this project, we used the machine learning framework PyTorch and Word2Vec word embeddings.

4.2.1. Experiments. In our first project, we used logistic regression with TF-IDF vectorization, achieving 80.47% accuracy. For this project, we implemented a neural network with Word2Vec embeddings, which yielded 75.84% accuracy on the validation set. Key experimental differences include:

- **Feature Representation:**
  - Project 1: TF-IDF (bag-of-words)
  - Project 2: Word2Vec embeddings (300-dimensional)
- **Model Architecture:**
  - Project 1: Linear logistic regression
  - Project 2: 2-layer neural network with ReLU activation and dropout
- **Hyperparameter Tuning:**
  - Project 1: Manual grid search
  - Project 2: Optuna automated optimization (30 trials)

Metric	Project 1 (Logistic Regression)	Project 2 (Neural Network)
Accuracy	80.47%	75.84%
Precision	80%	73.6%
Recall	81%	80.7%
F1 Score	80.5%	77%
Training Time	5 minutes	4 hours

4.2.2. Analysis. Here are some strengths and limitations of the neural network:

- **Strengths:**
  - Word2Vec captures semantic relationships better than TF-IDF
  - Neural networks can model non-linear decision boundaries
  - Dropout regularization helps prevent overfitting
- **Limitations:**
  - Requires significantly more computation time
  - More sensitive to hyperparameter choices
  - Harder to interpret than logistic regression

However, the neural network approach shows worse performance compared to logistic regression.
The difference in accuracy can be attributed to:

- Mean-pooling of Word2Vec embeddings, which may have led to information loss, weakening sentiment signal strength compared to TF-IDF's explicit term weighting.
- Retaining case sensitivity during Word2Vec lookup, which may have caused missed embeddings for capitalized words, whereas TF-IDF naturally handled such variations.
- TF-IDF's lexical weighting, which may have provided a better fit for sentiment classification in datasets where keyword presence is a strong indicator.

4.2.3. Plots.

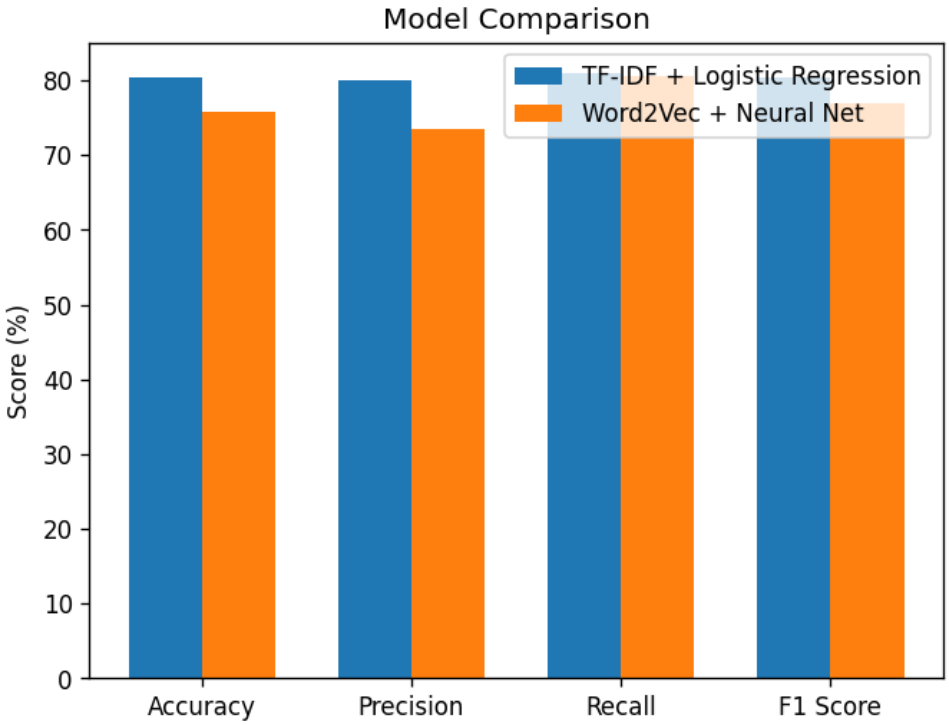

Figure 8: Performance comparison between Project 1 (blue) and Project 2 (orange) across metrics

Key observations from the plots:

4.2.4. Conclusion. Since the neural network approach underperforms the logistic regression model from Project 1 (75.8% vs. 80.4% accuracy), it demonstrates that traditional methods with hand-engineered features like TF-IDF can outperform deep learning when:

1. the dataset is relatively small (<200k samples),
2. sentiment relies heavily on specific keywords rather than complex linguistic patterns, and
3. computational resources favor simpler models.

For future work, we could make the following improvements:

1. Use pre-trained sentence embeddings (e.g., BERT) instead of averaged Word2Vec to better preserve phrase-level meaning,
2. Incorporate attention mechanisms to weight sentiment-bearing words more effectively, and
3. Apply case-insensitive embedding lookup to improve vocabulary coverage and reduce token mismatches.