UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *<first-name last-name>*
sdi: <sdiYYZZZZZ>

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

This project presents the development of a sentiment classification model for English-language tweets using deep neural networks and Word2Vec embeddings, implemented with the PyTorch framework. The approach focuses on leveraging pre-trained word vectors to capture semantic information and improve model generalization. Several deep learning architectures were explored, with a final model selected based on its performance across key evaluation metrics including accuracy, precision, recall, and F1-score.

2. Initial Approach and Baseline Model

As a starting point, a well-structured preprocessing function was applied to the specific tweet dataset in order to clean and normalize the data. The preprocessing steps included converting all text to lowercase, removing URLs, mentions, numbers, special and Unicode characters, and replacing emoticons and common social media expressions with sentiment tokens. Negation handling was also implemented to preserve semantic polarity. On the other hand, several preprocessing techniques were tested but eventually discarded due to negative or negligible impact on performance: lemmatization, stemming, spelling correction, stopword removal, and part-of-speech-based word filtering.

For word representation, the pre-trained GloVe word embeddings with 50 dimensions (`glove.6B.50d.txt`) were initially used. On top of that, a simple feedforward neural network was implemented, consisting exclusively of linear layers. Specifically, the model included a single linear layer `Linear(50 → 1)` with no non-linear activations, aiming to serve as a minimal but functional baseline for further experimentation.

This initial model showed signs of underfitting, likely due to the lack of non-linearity and limited model capacity. Nevertheless, it provided a useful reference point for comparison, achieving the following validation results:

- **Accuracy:** 0.674
- **Precision:** 0.664
- **Recall:** 0.707
- **F1 Score:** 0.685

3. 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.

3.1. Effective Preprocessing Techniques

- **URL Replacement with Placeholder Token**
  Tweets frequently include URLs, which typically do not carry meaningful sentiment information and are rarely repeated across different tweets. All URLs were replaced with a generic placeholder token `<url>`. This helped reduce vocabulary size and prevented the model from learning irrelevant patterns from unique or rare URL strings. By preserving only the structural information (i.e., that a link is present), the model achieved better generalization and improved performance on unseen data.

- **User Mention Replacement**
  Mentions (`@username`) were replaced with the token `<user>`, as usernames are unique and do not offer generalizable linguistic value. Treating them as regular tokens would lead to many out-of-vocabulary (OOV) cases and introduce noise. Replacing them with a consistent token allows the model to retain conversational context (direct replies) while avoiding sparsity and overfitting, contributing positively to model accuracy.

- **Number Replacement with Token**
  All numerical values were replaced with the token `<number>`. Numbers rarely provide sentiment unless heavily context-dependent, and the GloVe embeddings used in the model do not support most numeric combinations. By replacing them with a consistent token, we reduced the number of OOV tokens and improved vocabulary consistency. This normalization step contributed to a noticeable increase in both accuracy and training stability.

- **Repeated Punctuation Normalization**
  Repeated punctuation marks (`!!!`, `???`) were normalized to a single instance (e.g., `!`, `?`). While such repetitions may indicate emphasis, they are not supported by GloVe embeddings and lead to tokenization inconsistencies. Keeping a single instance preserved the emotional signal without introducing unnecessary vocabulary expansion. This preprocessing step improved model performance by reducing noise while still retaining expressive elements of punctuation.

- **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.

3.2. Ineffective Preprocessing Techniques

- **Negation Handling** Although negation is often considered crucial in sentiment analysis, in this case, incorporating techniques such as merging terms (e.g., converting not happy to not$_happy)didnotleadtoanyperformanceimprovement.Thisislikelybecausesu$
  $of-Vocabulary(OOV)terms.Furthermore,indatasetsconsistingofshort,informaltextliketweets,$

- **Lemmatization**
  Lemmatization 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. Although lemmatization slightly reduced accuracy so this is the reason why I excluded lemmatization preprocessing techniques.

- **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 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.

- **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" → "you" and "luv" → "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 not retained in the final pre-processing function.

- **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.

4. Word Embedding Experiments

At this stage, a series of experiments were conducted using different pretrained word embeddings, aiming to improve the semantic quality of the input text representations and, consequently, enhance the classifier's performance. Specifically, the following embeddings were evaluated:

GloVe.6B.200d: Transitioning from 50 to 200 dimensions significantly improved the model's ability to capture the semantic relationships between words, resulting in a noticeable performance increase. The validation accuracy reached approximately **0.719**, making this embedding a strong baseline for comparison.

Google News Word2Vec (300d): Although offering higher dimensionality (300 dimensions) and trained on a massive general-purpose corpus, its application to this task did not yield substantial improvements. In fact, the accuracy dropped slightly (estimated **Accuracy:** ∼**0.715**). This is likely due to the thematic and stylistic differences between news content and tweets, limiting the embedding's transferability.

GloVe Twitter 200d (glove.twitter.27B.200d): This embedding, trained specifically on Twitter data, proved to be the most effective. Its thematic and linguistic proximity to the task's dataset allowed the model to better understand abbreviations, slang, and special characters commonly found on Twitter. The validation accuracy reached **0.756**, making it the best-performing option among those tested.

Custom Word2Vec trained on the dataset: An attempt was also made to train a new Word2Vec model exclusively on the task-specific tweets, based on the assumption that it would better capture domain-specific semantics. However, the performance fell short compared to the pretrained embeddings (Accuracy: ∼**0.71**), likely due to the relatively small size of the dataset compared to the large corpora used to train existing models.

Conclusion: The `glove.twitter.27B.200d` embedding was ultimately selected for word representation, as it delivered the highest accuracy and demonstrated excellent adaptability to the characteristics of Twitter data, where informal language, emotional tone, and diverse linguistic patterns are prevalent.

5. Tokenization with TweetTokenizer

In order to prepare raw text data for embedding generation, tokenization is a crucial preprocessing step. For this project, we utilized the `TweetTokenizer` from the NLTK library. Unlike standard tokenizers, the `TweetTokenizer` is specifically designed for processing social media content such as tweets, which often contain irregular punctuation, emojis, hashtags, and user mentions.

`TweetTokenizer` provides more robust and context-aware token splitting compared to generic whitespace or word tokenizers. It helps preserve meaningful tokens while handling edge cases like contractions and repeated punctuation.

- Handles hashtags (e.g., `#example`) and emoticons as distinct tokens.
- Avoids splitting common patterns like "don't" or "can't" incorrectly.
- Removes unnecessary characters without discarding important semantic units.

This tokenizer significantly influenced the quality of the embeddings generated by ensuring that the input tokens passed to the word embedding model (Word2Vec or GloVe) were meaningful and properly segmented.

Finally, the use of TweetTokenizer led to an approximate 1 % increase in validation accuracy, highlighting its effectiveness in the context of noisy, informal text data.

6. Neural Network implementation

6.0.1. Overview. To develop a sentiment classifier, we implemented a fully connected feedforward neural network. The model consists of a configurable number of hidden layers, each followed by GELU activation functions. The output layer contains a single neuron without a sigmoid activation, since we use the BCEWithLogitsLoss function, which internally combines the sigmoid function with binary cross-entropy loss.

The forward pass is straightforward: each input sample is passed sequentially through a series of Linear layers, optionally followed by BatchNorm1d, GELU, and Dropout, depending on the configuration settings.

During model design, I experimented with the following hyperparameters:

- Activation Functions
- Hidden layer sizes (H1, H2)
- Number of layers
- Learning rate
- Batch size
- Dropout rate
- Use of Batch Normalization
- Optimizer selection (Adam, AdamW, RMSprop, Radam)
- Weight decay
- Epochs

Certain hyperparameters, such as the dropout rate, the number of training epochs, the shuffling of input samples, and other stochastic aspects of training (weight initialization), introduce variability into the learning process. This means that even with the same hyperparameter configuration, the final outcome can vary between runs. As a result, manual hyperparameter tuning becomes time-consuming and unreliable, making the use of automated optimization techniques, such as Optuna, essential.

In combination with the fact that the number of hyperparameters is large and their interactions can significantly affect model performance, Optuna was used for automated experimentation.

Optuna is a framework for hyperparameter optimization that leverages intelligent search techniques to automatically find the best set of parameters with a minimal number of trials, effectively balancing the need for extensive search with time and resource efficiency.

During the experimentation process, a targeted grid search was also conducted around the best results found by Optuna, as well as manual tuning attempts. However, manual tuning proved ineffective and was ultimately discarded.

6.0.2. Neural Network Prameters. After extensive experimentation with various architectural and training hyperparameters, the following configuration was found to perform best for our feedforward sentiment classifier:

- **Architecture (Model Design)**
  - **num_layers = 2**: For the number of hidden layers parameter, values of 2 and 3 were tested using Optuna across multiple runs. Although deeper networks theoretically offer increased representational capacity, in our experiments, models with two hidden layers consistently yielded better performance in terms of both accuracy and F1-score.

    This is likely due to the fact that the network already utilizes relatively large hidden dimensions (H1=344, H2=224), which provide sufficient capacity. Adding a third layer increased the model's complexity without offering substantial performance gains, and in some cases, led to convergence issues or tendencies toward overfitting.
  - **H1 = 344, H2 = 224**: The optimal hidden layer sizes (H1=344, H2=224) were selected via Optuna to balance model capacity and training cost. These dimensions are sufficient to capture complex linguistic patterns without excessive complexity, ensuring good generalization on the 150K-sample dataset. Smaller layers (64–224) lacked the capacity for language understanding, while larger ones (512+) led to overfitting and slower training.
  - **D_out = 1**: Since the task is binary classification a single output neuron is sufficient. This design returns a probability via sigmoid and simplifies training, which is why I didn't experiment with multiple outputs.
  - **Activation Function = GELU**: GELU was chosen over ReLU and ELU after experimentation, as it slightly reduced overfitting without slowing down training and without accurasy drop. Its smooth derivative contributed to more stable training, especially with large hidden layers (H1=344, H2=224). Compared to ReLU (better for small, fast models) and ELU (useful in shallow networks), GELU proved more suitable for deeper architectures and linguistically complex NLP tasks like sentiment analysis.
- **Training Parameters (Optimization Process)**
  - **optimizer = Adam**:Adam was selected as the optimizer after experimenting with AdamW, RAdam, and RMSprop. It demonstrated faster and more stable convergence at lower learning rates (lr = 0.000125), making it well-suited for the task. Although AdamW slightly reduced overfitting, it required higher learning rates to reach similar performance and still wasn't so good compared to Adam, as shown by the Optuna tuning results. Finally, RMSprop tended to increase overfitting, particularly with large hidden layers (H1=344, H2=224).
  - **learning_rate = 0.000125389...**: The learning rate of 0.000125 was selected through Optuna as it gave the best trade-off between accuracy and stable training. Higher values made the training unstable, while lower ones made learning too slow and caused underfitting. This relatively small learning rate also helped limit overfitting.

  - **scheduler = ReduceLROnPlateau**: I also used a ReduceLROnPlateau scheduler to automatically lower the learning rate when the validation loss stopped improving. This technique improved accurasy as well.
  - **batch_size = 128**: The batch size of 128 was selected through Optuna hyperparameter optimization as the optimal trade-off between training efficiency and gradient stability. Larger batch sizes (256) demonstrated inferior convergence, while smaller batches (32, 64) significantly increased training time without providing meaningful improvements in model performance.
  - **shuffle = True**: Enabling shuffle (True) ensures random sample ordering before each epoch, preventing the model from learning potential artifacts in the data sequence while improving generalization capabilities

- **Regularization Techniques (Generalization Control)**
  - **dropout = 0.3**:Drop out is one of the most efficient parameters. A dropout rate of 0.3 was selected as optimal for controlling overfitting while maintaining model accuracy. Without dropout, the model showed clear overfitting with high training but poor validation accuracy. The 30 % neuron deactivation during training forces the network to learn more robust features and reduces dependency on specific activation patterns, effectively creating an implicit ensemble effect particularly important for our large hidden layers (H1=344, H2=224). Testing showed higher rates (0.5) overly disrupted learning while lower rates (0.1) provided insufficient regularization. When combined with batch normalization, this 0.3 dropout rate improved validation accuracy by 2-3 % compared to non-dropout configurations, achieving the best balance between learning capability and generalization performance.
  - **weight_decay = 0.0**: The L2 regularization term penalizes large weight values to prevent overfitting. In our model, various weight decay values (e.g., 1e-4, 1e-5) were tested via Optuna but showed no accuracy improvement. The combination of dropout (p=0.3) and batch normalization already provided sufficient regularization, making L2 redundant for this architecture.
  - **batch_norm = True**: Batch normalization stabilizes training by normalizing each hidden layer's outputs and it accelerates convergence and reduces learning rate sensitivity. Compared to no batch norm, it improved validation accuracy by 1-2 %.

Trial	Accuracy	H1	H2	Layers	Optimizer	LR	Batch Size	Weight Decay	Dropout	Batch Norm
0	0.7919	152	128	3	Adam	2.20e-4	128	1e-6	0.30	True
1	0.7894	120	96	3	AdamW	1.29e-4	128	1e-6	0.10	True
3	0.7915	312	96	3	AdamW	7.13e-4	64	0.0	0.20	True
4	**0.7928**	376	224	2	Adam	3.00e-4	64	1e-4	0.40	True
8	0.7926	376	96	3	Adam	1.18e-4	128	1e-6	0.30	True

Table 1: Representative trials from Optuna hyperparameter tuning showing architecture, optimizer, learning rate, regularization, and accuracy.

7. Results and Overall Analysis

7.1. Results Analysis

The final model configuration demonstrated consistently strong performance across all key evaluation metrics. Using the GloVe Twitter 200-dimensional embeddings and a well-optimized feedforward neural network architecture, the classifier achieved a validation accuracy of approximately 0.789. The corresponding precision, recall, and F1-score values were also satisfactory, indicating that the model generalizes well and performs reliably across diverse examples in the validation set.

Compared to the initial baseline model, which reached an accuracy of 0.674, the improvements across all metrics highlight the success of the experimental strategy. This progress was largely due to more expressive embeddings, better preprocessing, and thorough hyperparameter optimization using the Optuna framework.

8. Evaluation

8.1. Model Evaluation Metrics

Accuracy.

- **Definition**: Ratio of correct predictions to total predictions
- **Interpretation**:
  - Training: 82.11% (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

8.2. Learning Curve

Figure 1 presents the learning curves for training and validation loss, as well as accuracy, across 27 epochs. These curves offer a comprehensive view of the model's learning behavior over time.

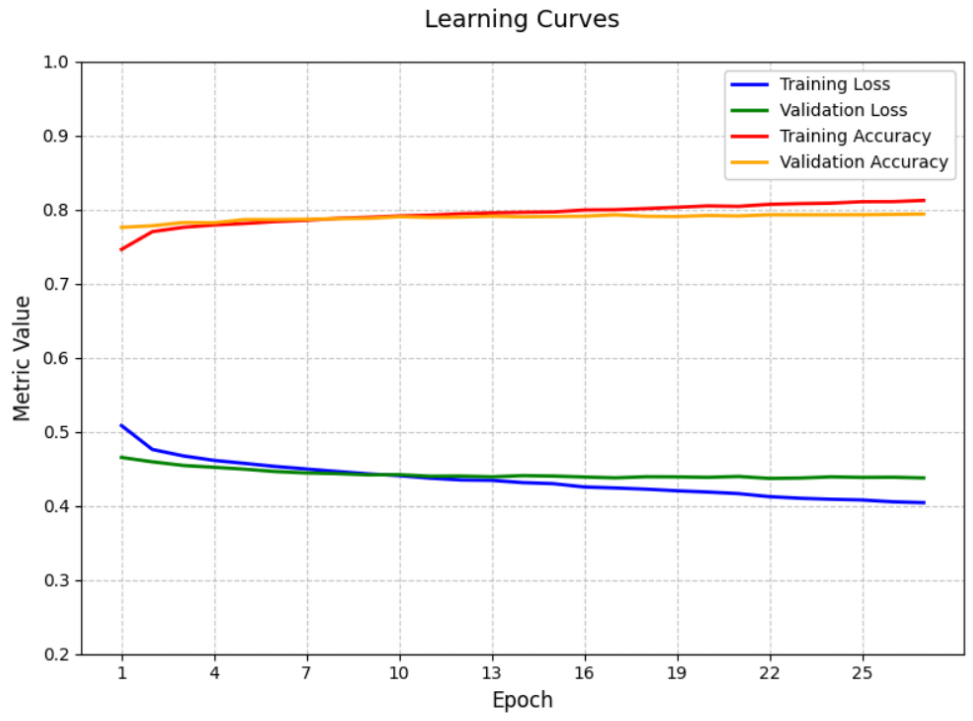

Figure 1: Learning curves: Loss and Accuracy for Training and Validation Sets

Training and Validation Loss: The training loss (blue line) steadily decreases throughout the epochs, showing that the model continues to learn effectively without stagnation. The validation loss (green line) also decreases, though at a slower rate, and begins to plateau around epoch 20. The absence of a significant divergence between the two curves suggests that the model does not overfit and maintains good generalization.

Training and Validation Accuracy: The training accuracy (red line) and validation accuracy (orange line) both increase gradually, converging to values around 0.80 and 0.79 respectively. The small gap between them indicates stable learning and well-regularized training. The performance plateau near the final epochs also suggests that the model has reached its optimal capacity under the given configuration.

Conclusion: The learning curves confirm the model's robustness. There is no indication of underfitting or overfitting, and the consistent trends across all four metrics reflect successful training dynamics and generalization performance.

8.3. ROC Curve

The ROC curve shown in Figure 2 illustrates the diagnostic ability of the classification model across different discrimination thresholds. The curve plots the *True Positive Rate (TPR)* against the *False Positive Rate (FPR)*. The closer the curve follows the top-left border of the plot, the better the model is at distinguishing between the positive and negative classes.

A key performance metric derived from the ROC curve is the Area Under the Curve (AUC). In this case, the model achieves an AUC of 0.88, which indicates strong overall classification performance. A value of 1.0 represents perfect classification, while 0.5 implies no discriminative power (i.e., random guessing). The dashed diagonal line serves as a baseline, representing the performance of a random classifier.

The shape of the ROC curve and the relatively high AUC suggest that the model maintains a good balance between sensitivity (recall) and specificity. This implies that the classifier is effective at minimizing both false negatives and false positives, making it suitable for tasks where accurate distinction between classes is critical.

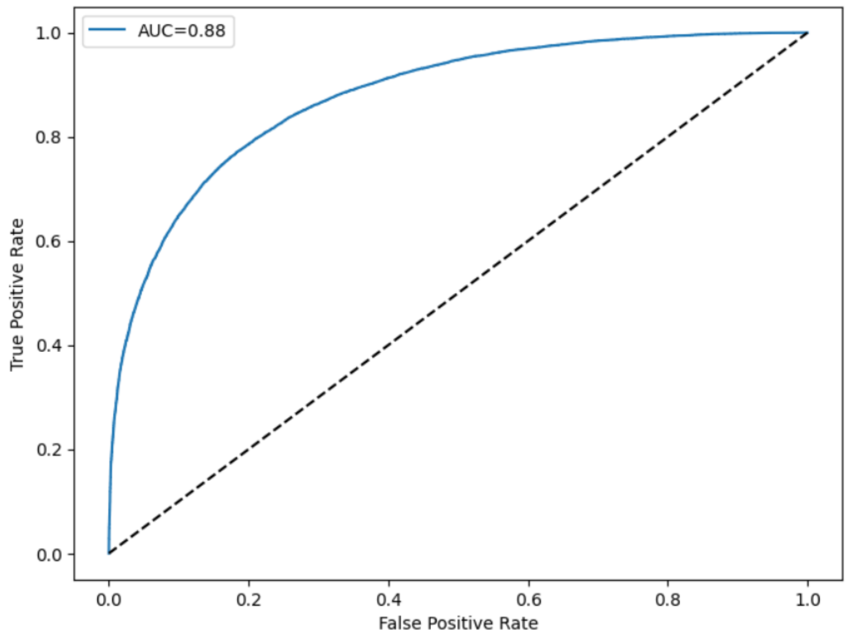

Figure 2: ROC Curve for the Classification Model (AUC = 0.88)

8.4. Comparison with the first project

Compared to the first project, which employed TF-IDF combined with Logistic Regression, the overall performance was slightly better in terms of stability and accuracy. The earlier model was deterministic, making it easier to reproduce results and interpret feature importance. However, the current project, which utilizes a neural network, offers greater modeling flexibility and the potential to capture more complex patterns in the data. While it introduces stochasticity and requires careful tuning, it demonstrates strong performance and scalability, especially for larger datasets or future extensions.

9. Bibliography

References

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

<You should link and cite whatever you used or "got inspired" from the web. Use the / cite command and add the paper/website/tutorial in refs.bib>
<Example of citing a source is like this:> [1] <More about bibtex>