Mohammed Abdullah Ebrahim 202106206

Yahya Zaqad 202110652

Saeed Al Ansari 202209377

Ghanim Masoud 202004603

Submitted on :Date: May 8, 2025

Sentiment Classification on IMDb Dataset Using Deep Learning:

1. Introduction

This project presents the development of a sentiment classification model for IMDb movie reviews. The team focused on comparing a deep learning model with traditional machine learning baselines and tuning the model for better performance.

2. Dataset and Preprocessing

The dataset used was the IMDb Movie Reviews, which includes 25,000 training and 25,000 testing samples. Each review is labeled as either 0 (negative) or 1 (positive).

For preprocessing, the vocabulary size was limited to the top 10,000 most frequent words to improve training speed and reduce noise. Reviews were padded or truncated to a length of 256 tokens to ensure uniform input size. Since the dataset is clean and pre-tokenized by Keras, no additional cleaning was needed. Data augmentation and normalization were not applied, as these are more suitable for image data; text data was handled with tokenization and padding.

3. Model Implementation

The deep learning model consisted of an embedding layer to convert words into dense vectors, followed by a Global Average Pooling 1D layer to average word vectors. A dense layer with ReLU activation was used to add non-linearity, and a final sigmoid output layer provided the binary classification result.

Hyperparameter tuning was done by testing different combinations of dense units (16 and 32), optimizers (Adam and RMSprop), and batch sizes (256 and 512). This resulted in 8 combinations, all tested manually through grid search. Early stopping was used to prevent overfitting and reduce training time. The best-performing configuration used 16 dense units, the Adam optimizer, and a batch size of 256.

To ensure reproducibility, all random seeds were set to 42 for NumPy, TensorFlow, and Python. The environment used included TensorFlow 2.18.0 and Python 3.9, running on Google Colab using a CPU.

4. Baseline Models and Exploring Other models

To evaluate the deep learning model, several traditional machine learning models were also tested. These included Logistic Regression using a bag-of-words representation, Naive Bayes (MultinomialNB), and a Decision Tree classifier.

These models were chosen for their simplicity and interpretability, serving as strong baselines for comparison. The neural network achieved an accuracy of 88.1%, while

Logistic Regression achieved 85.2%, Naive Bayes achieved 83.8%, and the Decision Tree achieved 78.6%.

5. Evaluation and Visualization

The evaluation metrics used included accuracy, precision, recall, F1-score, and the confusion matrix. These metrics provided a complete picture of model performance.

Visualizations included training and validation loss and accuracy curves, a confusion matrix heatmap, and a bar chart comparing the performance of all models. These visuals helped assess training dynamics and final model effectiveness.

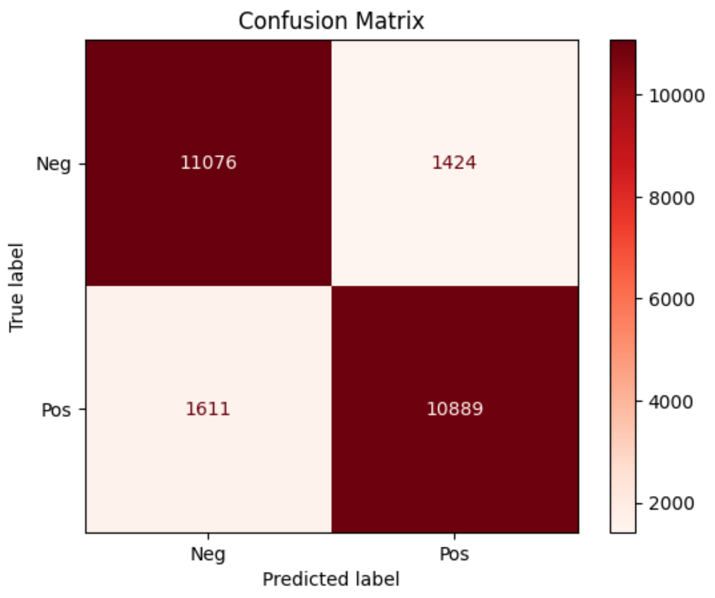

Figure 1 ConFusion Matrix

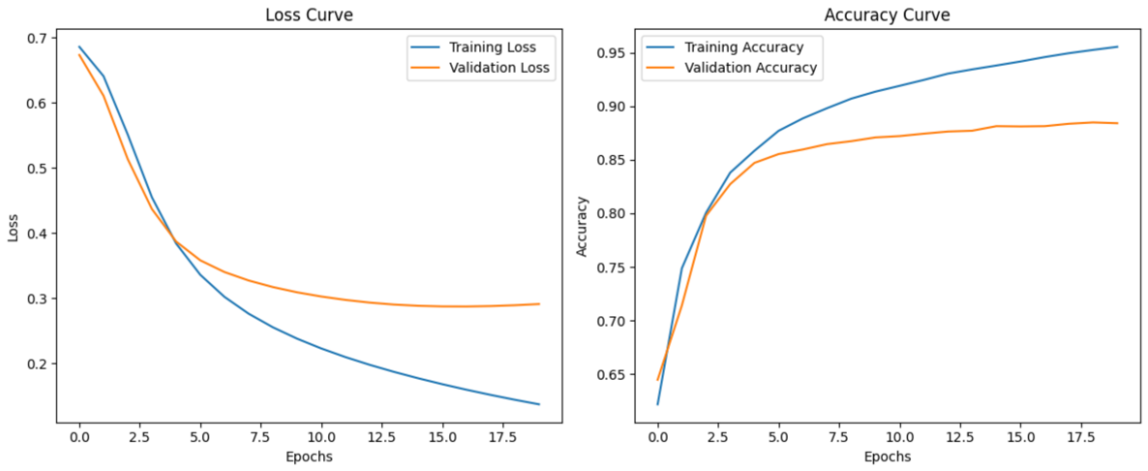

Figure 2 Loss Curve and Accuracy Curve

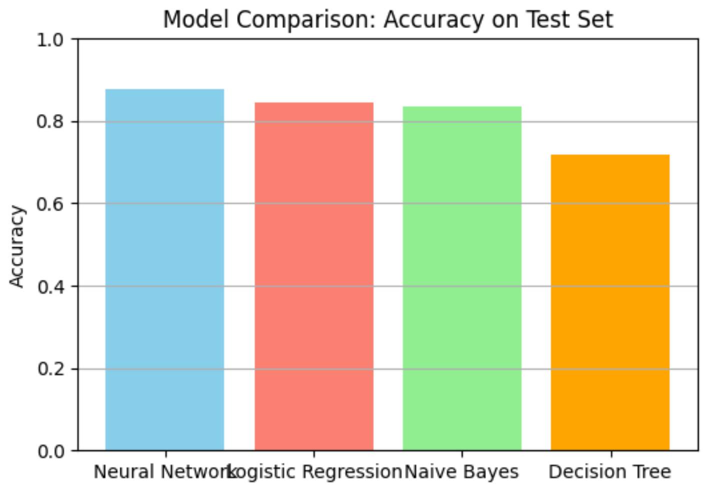

Figure 3 Comparison Between Our Model and the baseline model(Logistic Regression) and two other models

6. Testing with Custom Inputs

A feature was added to allow testing of the model on user-provided text inputs. This made it possible to observe the model's predictions on real, unseen sentences, demonstrating its practical value.

```
#testing the model on real input:
def review_to_sequence(review, word_index, maxlen=256):
   review = review.lower().split()
   encoded = [word_index.get(word, 2) + 3 for word in review if word in word_index]
   return pad_sequences([encoded], maxlen=maxlen)
```

```
word_index = imdb.get_word_index()

# Example 1
custom_review1 = "This movie was really amazing and I loved the acting!"
input_seq1 = review_to_sequence(custom_review1, word_index)
prediction1 = best_model.predict(input_seq1)[0][0]
print(f"\nReview 1: {custom_review1}")
print(f"Predicted Sentiment: {'Positive' if prediction1 > 0.5 else 'Negative'} (Confidence: {prediction1:.2f})")

# Example 2
custom_review2 = "The plot was boring and I fell asleep halfway through."
input_seq2 = review_to_sequence(custom_review2, word_index)
prediction2 = best_model.predict(input_seq2)[0][0]
print(f"\nReview 2: {custom_review2}")
print(f"Predicted Sentiment: {'Positive' if prediction2 > 0.5 else 'Negative'} (Confidence: {prediction2:.2f})")

# Example 3
custom_review3 = "It had some good moments, but overall it was disappointing."
input_seq3 = review_to_sequence(custom_review3, word_index)
prediction3 = best_model.predict(input_seq3)[0][0]
print(f"\nReview 3: {custom_review3}")
print(f"Predicted Sentiment: {'Positive' if prediction3 > 0.5 else 'Negative'} (Confidence: {prediction3:.2f})")

# Example 4
custom_review4 = "Absolutely fantastic! Best film I've seen in a while, Amazing"
input_seq4 = review_to_sequence(custom_review4, word_index)
prediction4 = best_model.predict(input_seq4)[0][0]
print(f"\nReview 4: {custom_review4}")
print(f"Predicted Sentiment: {'Positive' if prediction4 > 0.5 else 'Negative'} (Confidence: {prediction4:.2f})")
```

Test Results:

Review 1:

"This movie was really amazing and I loved the acting!"

Predicted Sentiment: Positive (Confidence: 0.71)

Review 2:

"The plot was boring and I fell asleep halfway through."

Predicted Sentiment: Negative (Confidence: 0.09)

Review 3:

"It had some good moments, but overall it was disappointing."

Predicted Sentiment: Positive (Confidence: 0.53)

Review 4:

"Absolutely fantastic! Best film I've seen in a while, Amazing"

Predicted Sentiment: Positive (Confidence: 0.71)

7. Conclusion

The project resulted in a successful sentiment classification model based on deep learning, with superior performance compared to baseline methods. The team ensured that all choices were justified, the implementation was reproducible, and the evaluation was thorough and well-documented.