***Sentiment Analysis Using Neural Network***

*WGU*

*Course Number: 604*

*Trayvonious Pendleton*

*011205284*

**A1: Research Question**

Can a neural network model accurately classify movie reviews as positive or negative using sentiment analysis techniques on the IMDb Sentiment Labelled Sentences dataset? This question is grounded in a real-world scenario: film studios, streaming platforms, and review aggregators often want to understand public sentiment toward content. We aim to automate this process using NLP and neural networks to support faster, data-driven content strategies.

**A2: Goals of the Analysis**

Thegoals of this analysis are:

**Sentiment Classification**:

* Build a model to categorize reviews into two sentiment classes: positive (1) and negative (0).

**NLP Pipeline Implementation**:

* Preprocess text and use word embeddings to represent text sequences.

**Model Training & Evaluation**:

* Train a neural network using a split of training, validation, and test data.
* Evaluate model performance using accuracy and loss metrics.

**Industry Relevance**:

* Show how this model could be used in production by a company like IMDb or Netflix to improve customer insights.

**A3: Industry Relevant Neural Network Model**

For this project, I selected a Long Short-Term Memory (LSTM) neural network to perform binary text classification on the IMDb Sentiment Labelled Sentences dataset.

LSTM is a type of recurrent neural network (RNN) that excels at handling sequential data and learning long-term dependencies, which is critical in understanding the sentiment expressed in natural language. LSTMs are designed to overcome the vanishing gradient problem in standard RNNs and are widely used in the industry for NLP tasks, including:

* Sentiment analysis
* Text classification
* Named entity recognition
* Language modeling

**Justification for LSTM in this Project:**

* The IMDb dataset contains movie reviews, where the order of words impacts sentiment. LSTM is well-suited for learning from such sequences.
* Compared to more complex architectures like BERT, LSTM offers faster training and lower computational requirements, which is appropriate given the relatively small dataset.
* LSTM can learn context over multiple words, which is necessary for capturing nuanced expressions like sarcasm or negation (e.g., “not a bad movie”).

**Industry Relevance:**

* LSTM-based architectures are used in many commercial sentiment analysis tools and chatbot applications. They provide an outstanding balance of accuracy, efficiency, and interpretability for medium-sized NLP tasks.

**B1: Exploratory Data Analysis**

Exploratory data analysis is crucial in preparing text data for training a neural network. It provides insights into the data’s structure, quality, and distribution, guiding key preprocessing decisions like vocabulary size, padding, and embedding dimensions.

**Presence of Unusual Characters:**

Unusual characters such as emojis, foreign letters, or special symbols can confuse the tokenizer and negatively impact model performance. This step scans all cleaned sentences for such characters to ensure the dataset contains only standard English words and punctuation-free content.

Result:

* Total unusual characters found: 0

This confirms that the dataset is clean and free of problematic characters. No additional filtering was necessary before tokenization.

**Vocabulary Size:**

Vocabulary size refers to the number of unique words present in the dataset. This information is used to configure the embedding layer and to help prevent the model from learning noise or overfitting rare words. The tokenizer assigns a unique integer to each word.

Result:

* Vocabulary size: 3,174 unique tokens

This is a manageable size for a small dataset like IMDb labelled sentences, and it’s well-suited for a lightweight LSTM model.

**Sentence Length Statistics:**

LSTM models require input sequences of consistent length. We need to analyze sentence length distribution to choose an appropriate padding length. Statistical measures like average, standard deviation, and percentiles help identify a suitable cutoff length that balances model performance and memory efficiency.

Results:

* Average sentence length: 19.44 words
* Standard deviation: 67.12
* Minimum length: 1 word
* Maximum length: 1,384 words
* 95th percentile length: 34 words

The 95th percentile value of 34 is selected as the future padding length. This means 95% of all sentences are 34 words or shorter, minimizing the risk of truncating important data while avoiding excessive padding.

**Selected Embedding Length:**

Each word will be mapped to a dense vector in the embedding layer. The embedding size determines the dimensionality of these vectors. A 100-dimensional embedding is a common industry standard for small to medium NLP tasks. It provides enough capacity to capture meaningful word relationships without adding unnecessary computational overhead.

Result:

* Selected embedding size: 100 dimensions

The dataset is clean and statistically well-behaved. The key insights gained from this analysis are:

* There is no need for special character filtering.
* A vocabulary size of 3,174 is appropriate for modeling.
* A padding length of 34 tokens captures 95% of sentences.
* An embedding size of 100 dimensions offers a strong balance of expressiveness and efficiency.

**B2: Tokenization Process**

Tokenization is converting raw text into a sequence of integers where each unique word is assigned a numeric ID. This step is essential because neural networks operate on numbers, not text.

**Why It’s Important:**

* It makes text understandable by the model.
* Creates a consistent numeric representation of each review.
* Handles out-of-vocabulary words with a special token, avoiding crashes or random behavior during testing.

**Steps and Code:**

**Step 1: Initialize the Tokenizer**

from tensorflow.keras.preprocessing.text import Tokenizer

# Create tokenizer and fit on cleaned sentences

tokenizer = Tokenizer(oov\_token="<OOV>")

tokenizer.fit\_on\_texts(df["cleaned\_sentence"])

**Step 2: View Vocabulary**

word\_index = tokenizer.word\_index

print(f"Number of unique words (vocab size): {len(word\_index) + 1}") # +1 for padding token

**Step 3: Convert Sentences to Sequences**

sequences = tokenizer.texts\_to\_sequences(df["cleaned\_sentence"])

print("Example sequence (first review):")

print(sequences[0])

**Results:**

* Vocabulary size: 3,174
* Example sequence (first review): [3, 29, 29, 29, 1181, 1182, 13, 37, 3, 1183, 1184, 395, 141]

**B3: Padding Process**

**A graph with numbers and a line

AI-generated content may be incorrect.**

All sequences must have the same length to prepare the text data for input into an LSTM model. This is because neural networks require fixed-size input matrices for efficient computation and batching.

Padding is the process used to standardize the length of sequences. In this case, I applied post-padding, which means that zero values (0) are added after the original tokenized sentence until the desired length is reached. This preserves the original word order and ensures that meaningful tokens are not shifted or truncated at the beginning of the sequence.

Based on exploratory data analysis (B1), the maximum sequence length was set to 34 words, representing the 95th percentile of all sequence lengths in the dataset. This length captures most sentence structures while reducing excessive padding from more extended sequences.

The padding was implemented using the pad\_sequences function from TensorFlow’s Keras API with the following parameters:

* pad\_sequences(sequences, maxlen=34, padding='post')

**Summary of Padding Strategy:**

* Padding type: Post-padding
* Sequence length after padding: 34 tokens
* Padding values: 0
* Padding Location: After the original token sequence

A single padded sequence was visualized using Matplotlib to demonstrate the result. The visualization shows real tokens in lighter shades and padding (zeros) in darker tones. The saved image (padded\_sequence\_visual.png) is included in the submission as a screenshot of one padded sequence.

**B4. Sentiment Categories and Activation Function**

The IMDb sentiment dataset used in this project contains two sentiment labels:

* 0 for negative sentiment
* 1 for positive sentiment

This is a binary classification task, which means the model should output one value that can be interpreted as the likelihood of a review being positive.

I selected the sigmoid activation function for the final dense layer in the neural network, which is standard practice for binary classification problems. The sigmoid function outputs values between 0 and 1, allowing for probabilistic interpretation of sentiment. This supports using a simple threshold to determine whether the model predicts positive or negative sentiment.

Number of sentiment categories: 2

Activation function for the output layer: sigmoid

**B5: Explanation of Data Preparation and Dataset Splitting**

To prepare the IMDb Sentiment Labelled Sentences dataset for analysis using an LSTM model, several key steps were followed to clean and structure the data for optimal performance. Each step plays a critical role in shaping the final input used by the neural network.

**Data Cleaning:**

The raw sentences were first cleaned to ensure consistent and noise-free text input. This included:

* Converting all text to lowercase
* Removing all punctuation and special characters
* Eliminating extra whitespace

This step ensured that the tokenizer would focus only on the essential components of the text (words), preventing misleading tokens caused by inconsistent formatting.

**Tokenization:**

Using TensorFlow’s Tokenizer, each sentence was converted into a sequence of integers. Each word in the cleaned dataset was assigned a unique token ID based on frequency. An oov\_token (””) was also specified to handle any out-of-vocabulary words during inference.

**Total vocabulary size**: 3,174 unique tokens

**Sequence Padding:**

Since neural networks require fixed-length input sequences, all tokenized sequences were padded to a maximum length of 34 words, corresponding to the dataset's 95th percentile of sentence lengths. This ensures that 95% of the reviews fit without truncation, minimizing information loss while preserving memory.

Padding type: Post-padding (zeros added at the end)

Maximum length after padding: 34 tokens

**Label Preparation:**

Each sentence was associated with a label:

* 0 = negative sentiment
* 1 = positive sentiment

These labels were extracted and converted into a NumPy array to match the shape of the input features.

**Dataset Splitting:**

To ensure the model is trained and evaluated fairly, the dataset was split into training, validation, and test sets following industry-standard best practices:

* Training set (70%): used to train the model
* Validation set (15%): used to tune hyperparameters and monitor performance during training
* Test set (15%): used to evaluate final model performance on unseen data

**B6: Save the Prepared Dataset**

**A white rectangle with green text

AI-generated content may be incorrect.**

**C1: Output of the Model Summary**

**A screenshot of a computer

AI-generated content may be incorrect.**

**C2: Layers, Types, and Parameter Counts**

The model contains three main layers, each serving a unique function in the text classification pipeline:

**Embedding Layer:**

* Type: Embedding (input\_dim=3174, output\_dim=100, input\_length=34)
* Function: Converts each word index into a dense vector of size 100. This layer enables the model to learn semantic representations of words.
* Parameters: 3174 (vocabulary size) × 100 (embedding dim) = 317,400 parameters

**LSTM Layer:**

* Type: LSTM (64)
* Function: Processes the sequence of embeddings to capture word dependencies and temporal patterns in the text.
* Parameters: 42,240 trainable parameters

**Dense Output Layer:**

* Type: Dense (1, activation='sigmoid')
* Function: Outputs a probability indicating the sentiment classification (positive or negative).
* Parameters: 64 (input) + 1 (bias) = 65 parameters

**Total Parameters: 359,705:**

* **Trainable Parameters**: 359,705
* **Non-Trainable Parameters**: 0

All parameters are trainable, allowing the model to learn word representations and classification logic directly from the data.

**C3: Justification of Hyperparameters**

The following hyperparameters were selected based on industry standards for binary text classification tasks using LSTM models:

**Activation Functions:**

* Hidden Layer (LSTM): Internally uses the default tanh and sigmoid activations.

Tanh helps maintain stable gradients, while sigmoid gates control input, output, and forget mechanisms in the LSTM cell.

* Output Layer: The sigmoid activation function was chosen to output a probability between 0 and 1 for binary classification. This allows a clear threshold to classify reviews as positive or negative.

**Number of Nodes per Layer:**

* Embedding Layer: 100 dimensions were used to capture rich semantic features from words.
* LSTM Layer: 64 units were selected to provide a balance between model complexity and training efficiency.
* Dense Layer: A single node was used to output the binary classification decision.

**Loss Function:**

* Binary Crossentropy was chosen because the model solves a binary classification problem. This loss function measures the difference between the predicted probabilities and actual labels, optimizing the model for accuracy**.**

**Optimizer**

* The Adam optimizer was selected for its ability to adapt the learning rate during training, resulting in faster convergence and better performance on small to medium-sized datasets.

**Stopping Criteria**

* The training was set to run for up to 50 epochs, but early stopping was applied to monitor the validation loss. Training was halted if no improvement was observed for three consecutive epochs, and the best weights were restored.
* This prevents overfitting and ensures efficient use of computational resources.

These hyperparameters were chosen based on best practices in NLP modeling and were validated through experimentation with the dataset to ensure effective performance.

**D1: Model Training Process and Stopping Criteria**

The model was trained using a maximum of 50 epochs, with early stopping enabled to monitor validation loss. The stopping criteria were triggered after Epoch 7 when the validation loss failed to improve over three consecutive epochs.

Best Epoch: Epoch 4

Validation Accuracy: 0.7321

Validation Loss: 0.5701

After this point, the model began overfitting — validation loss increased despite continued improvements in training loss.

The training process is illustrated in the loss curve plot below:

**A graph with a line graph

AI-generated content may be incorrect.**

**D2: Assessment of Underfitting or Overfitting**

Based on the model’s training history and the final training plot, there is clear evidence of overfitting.

* Both training and validation loss decreased from Epoch 1 to Epoch 4, indicating the model was learning valuable patterns.
* After Epoch 4, validation loss began to increase, while training loss decreased sharply.
* By Epoch 6, training loss was down to approximately 0.25, while validation loss had risen above 0.90.

This divergence between training and validation loss indicates that the model was starting to memorize the training data rather than generalize well to unseen validation data.

**Summary:**

* Training loss: Continues decreasing across epochs (down to 0.25)
* Validation loss: Initially improves, then increases (peaks around 0.97)
* Conclusion: The model is overfitting after Epoch 4

Early stopping addressed this, and training was halted to prevent further overfitting. The model’s best weights were restored from Epoch 4, where validation performance peaked.

**D3: Visualization**

**Training and Validation loss values**

**A graph with blue and orange lines

AI-generated content may be incorrect.Training and Validation Accuracy Values**

**A graph with a line

AI-generated content may be incorrect.**

**D4: Predictive Accuracy of the Trained Model**

After training, the model was evaluated on a separate test set to assess its ability to generalize to new, unseen data. The evaluation metric used was accuracy, which measures the proportion of correct predictions from the total predictions.

**Test Set Evaluation Result (from D3):**

* Test Accuracy: 70.80%
* Test Loss: 0.5950

This means the model correctly classified approximately 71% of the movie reviews in the test set. Given that this is a binary classification task (positive vs negative sentiment), an accuracy above 70% indicates that the model has successfully learned meaningful patterns from the training data.

**Interpretation:**

* The model performs significantly better than random guessing (50% baseline).
* While not perfect, this level of accuracy is acceptable for a baseline LSTM on a relatively small dataset.
* The model shows potential for improvement through techniques like regularization, larger datasets, or fine-tuning of hyperparameters.

**Visual Support:**

Two training process-visualizations were generated:

* Training and validation loss over epochs
* Training and validation accuracy over epochs

These plots confirm the model’s ability to fit the training data and reveal signs of overfitting after Epoch 4, which was addressed using early stopping. Despite that, the validation accuracy remained stable, and test accuracy remained strong, supporting the model’s predictive capability.

**D5: Explanation of Ethical Compliance and Bias Mitigation**

This analysis complies with global ethical standards for artificial intelligence by promoting fairness, transparency, and responsible use of data. The project addresses key ethical concerns related to AI and machine learning systems, including the risk of bias, data integrity, and model explainability.

**Compliance with AI Ethical Standards:**

* **Transparency**: Visualizations and metrics have documented and supported the entire process — from data preparation to model training and evaluation. The logic behind model design and hyperparameter choices is clearly explained.
* **Accountability**: The model’s performance was evaluated using a separate test set, with early stopping used to prevent overfitting and to ensure the model does not make decisions based on memorization or noise.
* **Data Privacy & Safety**: The IMDb dataset contains public, anonymized movie reviews. No personally identifiable information (PII) was used in this analysis.
* **Inclusivity**: The model was trained and evaluated using a dataset with various language patterns. Care was taken to tokenize, clean, and balance the text to minimize linguistic bias.

**Bias Mitigation Steps:**

* **Balanced Sentiment Labels**: The dataset was manually labeled with an equal or near-equal number of positive and negative reviews, which helps prevent the model from being biased toward a particular class.
* **Early Stopping**: Used to stop training at the optimal point, preventing overfitting to specific samples that may introduce skewed behavior.
* **Generalization Validation**: The test set performance (70.80% accuracy) shows that the model generalizes reasonably well to new data rather than overfitting a specific review style or sentiment pattern.

This project demonstrates that machine learning can be developed in a way that respects both technical goals and ethical boundaries. The model was built and evaluated responsibly, carefully considering transparency, bias, fairness, and trustworthiness.

**E: Code to Save the Trained Model**

A blue line on a white background

AI-generated content may be incorrect.

**F. Functionality of the Model & Impact of Network Architecture**

The model is designed to perform binary sentiment classification, predicting whether a movie review expresses a positive or negative sentiment.

The model architecture consisted of:

* An Embedding layer that converted words into 100-dimensional vector representations.
* A Long Short-Term Memory (LSTM) layer with 64 units allowed the model to learn patterns across sequences of words and capture context over time.
* A Dense output layer with a sigmoid activation function to output a probability score.

**Impact of LSTM:**

* The LSTM architecture was crucial in capturing the order of words and their dependencies, making it more effective than simpler models like MLPs.
* It outperformed chance-level guessing (50%) and reached ~70.8% accuracy, proving that the model was functional and could extract useful insights from textual data.
* Although overfitting occurred, early stopping helped preserve the best-performing weights.

**G: Recommendation Based on Research Results**

The original research question was:

*“Can a neural network model accurately classify movie reviews as positive or negative using sentiment analysis techniques?”*

Based on the model’s performance and test accuracy of 70.80%, we can conclude that the answer is yes — a neural network, specifically an LSTM-based architecture, can successfully classify sentiment from short movie reviews with a reasonable level of accuracy.

**Recommendation:**

* The model can be used as a baseline sentiment analysis tool in production.
* For better results, it is recommended to:
* Expand the dataset (more reviews = better generalization)
* Experiment with Bidirectional LSTM or Transformer-based models like BERT
* Fine-tune hyperparameters and introduce dropout to reduce overfitting
* Deploy the model behind a simple API to automate review analysis for film studios, streaming platforms, or user feedback systems.

**J: Citations**

Kotzias, D. (2015). Sentiment Labelled Sentences [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C57604.