Laycee Glass

NLM3 Task 2

Advanced Data Analytics

4/19/25

Western Governors University

## NLM3 Performance Assessment, Task 2

### Student Information

Student Name: Laycee Glass

ID Number: 001896026

Date: 4/19/25

### Part I: Research Question

### A1. Research Question

The research question used for this assignment was the following: Can future Yelp reviews be accurately classified as having a positive or negative sentiment based on patterns learned from past customer reviews using neural network models?

### A2. Goals/Objectives

The objective of this analysis was to build a neural network model that can predict whether a customer would leave a positive or negative review based on language used in past reviews. Understanding patterns in word usage will help the restaurant identify trends and areas for improvement, guiding decisions to enhance customer satisfaction.

### A3. Neural Network

For the task at hand, the neural network chosen was a recurrent neural network (RNN). It was constructed and trained on the text sequences from the Yelp dataset to make meaningful predictions. Unlike other neural network types, RNNs are specifically designed for sequential data. This makes it ideal for our purposes – handling the ordered nature of words in customer reviews, and it allows the model to capture contextual relationships across word sequences, which is crucial for sentiment classification (Ibm, 2025).

To enhance the RNN’s performance, a long short-term memory (LSTM) layer was incorporated as well. LSTMs are well-suited for text classifications as they are designed to help “remember” useful information for longer periods. This means the model can keep track of key details in the reviews and make better predictions on their sentiment category (GeeksforGeeks, 2023).

### Part II: Data Preparation

### B1. Exploratory Data Analysis

To examine the dataset for unusual characters, a custom loop was written to iterate through each character in the ‘reviews’ texts. This process helped identify non-standard characters such as punctuation marks, accented letters, symbols, and numbers that may not contribute meaningfully to sentiment analysis.

A white background with black text

Description automatically generatedAfter identifying these characters, a regular expression-based cleaning step removed them from the dataset. The unusual characters were replaced with a space to ensure only standard alphabetical characters were retained for future processing.

Additionally, common stopwords were identified and removed using natural language toolkit (nltk)’s predefined list of stopwords. These words do not contribute significantly to contextual relationships or sentiment. As such, they were excluded to ensure the model focused on more meaningful words that contribute more directly to classification.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

To determine vocabulary size of the dataset, a tokenizer from the keras library was utilized. The tokenizer scanned through all the review in the dataset and assigned each unique word a corresponding integer. This created a word index which captured the total number of unique words in the text.

A screenshot of a computer code

Description automatically generatedThe vocabulary size was specifically calculated by finding the length of this word index and adding 1 to account for a padding token. This final count of 1769 represented the total number distinct tokens that the model accounted for when processing the text.

A word embedding length of 64 was chosen for the embedding layer of the neural network. It was selected as it provided a balance between model complexity and ability to capture relationships in the data. A larger embedding length might increase the model’s capacity to capture nuances in the data but could also lead to more overfitting. On the other hand, a smaller embedding length could lead to a diminished ability for the model to capture complex relationships within the data. By using 64, the model was able to efficiently learn meaningful representations of words while remaining computationally feasible for training.

A screen shot of a computer code

Description automatically generatedTo determine an appropriate maximum sequence length for the reviews, I calculated the length of each review after tokenization. I focused on the maximum, average, and 95th percentile lengths across the dataset. The results showed that while some reviews were longer, 95% had 12 words or fewer. In turn, a maximum sequence length of 12 was selected. This length struck a balance between including the majority of reviews without introducing excessive padding.

### B2. Tokenization Goals

The goal of tokenization was to convert human-readable text into a machine-readable form by breaking down each review into tokens. This is a critical step because these models operate on numerical data rather than raw text.

I used the tokenizer class from TensorFlow’s keras library to achieve this (seen above in section B1). The tokenizer assigned a unique integer to each distinct word in the dataset, effectively building a vocabulary that could be used to transform the text into sequences of integers. Before tokenization, I normalized the text in several ways to ensure consistency:

* Lowercasing: All characters were converted to lowercase so words would be treated the same.
* Removing punctuation: Characters not contributing to sentiment were stripped.
* Lemmatization: I used WordNet Lemmatizer from the nltk library to reduce words to their root form, helping group similar words under a common representation.
* Stopword removal: Words not contributing to sentiment were removed, based on a predefined list.

This sequence of preprocessing steps enhanced the quality of the input and ensured that the model was trained on cleaner and clearer data (GeeksforGeeks, 2024).

### B3. Padding

To standardize the length of all input for the RNN, I applied padding to make sure each review contained the same number of tokens. Padding was necessary because neural networks require fixed-size input, but Yelp reviews vary in length.

A screenshot of a phone

Description automatically generatedA screenshot of a computer program

Description automatically generated Based on my earlier explorations, I selected a maximum sequence length of 12 words. I used the pad\_sequences function to apply pre-padding. This means that zeros were added before the actual tokens in each sequence that shorter than 12 words. An example of this can be seen below.

In this sequence, the integers (107, 1396, 175, etc.) represented specific words from the original review based on the tokenizer’s word index. Since the review only had 7 tokens, the first 5 positions were filled with zeros to reach the maximum length. These zeros were simply placeholders to maintain consistent input size.

### B4. Sentiment Categories

The model classified reviews into two sentiment categories: positive (1) and negative (0). Since this was a binary classification, the final dense layer of the neural network used the sigmoid activation function. This outputted a probability score between 0 and 1, which indicated the likelihood that a given review was positive.

### B5. Data Preparation Steps

The data preparation process involved multiple steps, most of which have already been discussed but will once again be reviewed here. These were as follows: removing unusual characters, converting text to lowercase, removing punctuation, lemmatization, removing stopwords, tokenization, and padding.

Next, the data was split into training and testing sets using an industry-standard ratio of 80/20 (this code can be seen in section B3). This allows for the data to be trained on a large subset while preserving a portion for evaluation on unseen inputs. Specifically, the training set contained 800 samples, and the testing set contained 200 samples.

A screenshot of a computer code

Description automatically generated

During model training, an additional 30% of the training data was used as a validation set (can be seen in section C2). This set was used to monitor model performance during training, helping to tune hyperparameters and apply early stopping to prevent overfitting.

### B6. Copy of Cleaned Data Set

A copy of the cleaned data set was exported to its csv file for submission alongside this written report.

A screenshot of a computer code

Description automatically generated

### Part III: Network Architecture

### C1. Model Summary

The final neural network architecture used for sentiment classification is summarized below.

A screenshot of a computer

Description automatically generated

### C2. Model Layers

The model consisted of 5 layers in total:

1. Embedding layer: Converted each word into a 64-dimensional dense vector, enabling the model to learn the semantic relationships between words

* Output shape: (None, 12, 64)
* Parameters: 113,216

1. LSTM layer: Captured temporal dependencies and the order of words within the review sequences

* Output shape: (None, 128)
* Parameters: 98,816

1. Dense layer 1: Fully connected layer with 100 neurons to process features extracted by the LSTM

* Parameters: 12,900

1. Dense layer 2: Another dense layer with 50 neurons to reduce dimensionality and help in feature selection

* Parameters: 5,050

1. Output dense layer: A single-neuron layer using a sigmoid activation function to classify sentiment as either positive (1) or negative (0)

* Parameters: 51

These parameters were learned during training through back propagation. The trainable parameters were updated through backpropagation, while non-trainable parameters remained constant.

### C3. Hyperparameters

Rectified linear unit (ReLU) was chosen for the hidden layers because it allowed the model to learn complex patterns while remaining computationally efficient. Sigmoid was used in the final layer to provide a probability score for binary classification.

100 and 50 nodes were selected for the two inner dense layers to provide the model with enough capacity to learn from the extracted features. The final layer had 1 node with sigmoid activation for binary output. For a multi-class classification problem, the ‘softmax’ activation function would have been used instead of sigmoid.

Binary crossentropy was used as the loss function because it was appropriate for binary classification, which we have heavily established was our case. It effectively penalized incorrect predictions based on probabilities.

The ‘adam’ optimizer was chosen for its adaptive learning rate and reliable convergence performance, making it ideal for text-based tasks.

Early stopping was implemented to monitor validation loss. If the loss did not improve after 4 consecutive epochs (patience = 4), training was stopped to avoid overfitting.

A screenshot of a computer program

Description automatically generated Accuracy was used as the primary evaluation metric because it provided an intuitive measure of how often the model correctly predicted sentiment labels. All of these parameters can be seen in the code below.

### Part IV: Model Evaluation

### D1. Stopping Criteria

To prevent overfitting, I implemented early stopping using keras’ EarlyStopping callback. This monitored the validation loss and stopped training if the model failed to improve after 4 consecutive epochs.

Although I did not use restore\_best\_weights = True, the early stopping mechanisms still helped halt unnecessary training once the model plateaued. This improved efficiency and prevented excessive overfitting.

A screenshot of a computer

Description automatically generated The model was allowed to train for a maximum of 30 epochs, but training stopped earlier based on validation performance. Below is a screenshot showing the output of the final epoch before early stopping was triggered.

### D2. Model Fitness

I reviewed both the training versus validation accuracy and training versus validation loss graphs (seen below) to assess fitness of the model. The accuracy graph showed training accuracy improved steadily across epochs, while validation accuracy plateaued around epoch 5. Similarly, the loss graph showed training loss decreased drastically while validation loss began increasing after several epochs. These patterns suggested that the model was overfitting to the training data.

A screen shot of a computer program

Description automatically generated

A graph with blue and orange lines

Description automatically generatedA graph of a graph with numbers and lines

Description automatically generated with medium confidence

Once again, I used early stopping with a patience of 4 to address this. This halted the process once training and validation performance was no longer improving. I also used dropout in the LSTM layer, both input and recurrent to reduce overfitting by preventing the model from memorizing exact patterns in the training data. This helped the model learn more general features instead.

### D3. Model Visualizations

The graphs have already been seen and discussed in previous section (D2). To reiterate, training accuracy steadily increased while validation accuracy plateaued. Training loss decreased as validation loss began to rise. These trends suggested overfitting, which was addressed be early stopping and dropout.

### D4. Predictive Accuracy

The predictive accuracy of the trained network was evaluated using validation accuracy and a confusion matrix. The final validation accuracy achieved was approximately 76.25%, indicating that the model correctly classified sentiment for about three-fourths of the unseen reviews.

After training, the model was further evaluated by the confusion matrix below. This matrix showed that the model correctly classified 73 negative reviews and 71 positive reviews, while incorrectly predicting 27 negative reviews as positive and 29 positive reviews as negative.

### A screenshot of a graph Description automatically generated

Based on the matrix, we can calculate the following metrics:

* Accuracy =
* Precision =
* Recall =
* F1 Score =

These results suggest that the model performed reasonably well in distinguishing between positive and negative sentiments. An overall accuracy score of 72% means our model will correctly predict sentiment about 7 out of every 10 times.

The precision score showed the model is moderately effective at minimizing false positives, and recall showed the model struggled with catching all true positives. The F1 score reinforced our conclusion that the model was imperfectly consistent across both types of classification errors. This was appropriate for a first iteration model trained on a smaller dataset with limitations.

### Part V: Summary and Recommendations

### E. Saving Trained Network

The code used to save the trained network can be seen here:



### F. Functionality and Impact

The neural network for this assignment was designed to classify customer reviews into positive and negative sentiment categories based on content of the text. The network’s functionality relied on sequential processing of words, essential for understanding review context.

The architecture, as previously discussed, included:

* An embedding layer to transform words into dense vectors that capture semantic meaning
* An LSTM layer to capture sequential relationships and remember important patterns across word sequences
* Dense layers to progressively reduce dimensionality and extract relevant features for final classification

This architecture allowed the model to learn both the meaning of individual words and the contextual flow of sentences – both of which were important for correctly interpreting customer sentiment. The LSTM layer, in particular, played a key role by preserving word order and memory across the sequences, which improved the model’s ability to differentiate between subtle positive and negative expressions. Dropout was incorporated to reduce overfitting, alongside early stopping.

This, overall, helped the model achieve an accuracy of 76.25%, indicating it captured the general patterns necessary for classification while also managing model complexity.

### G. Course of Action

Based on the model’s results, I recommend that the restaurant should not rely solely on this model to predict customer sentiment. Although the model achieved a reasonable validation accuracy, it still misclassified a notable portion of reviews as seen in the confusion matrix. Therefore, the model should be treated as a supporting tool rather than a definitive decision-maker.

To improve reliability, I recommend the following actions for the future: collecting more data, refining text preprocessing, experimenting with different models, and regular retraining of the model.

Using a larger and more diverse set of customer reviews could help the model generalize better. More advanced techniques like n-grams, stemming, or even named entity recognition (NER) could enhance the model’s understanding of context. Trying alternative architectures, such as bidirectional LSTM and gated recurrent unit (GRU) layers, could significantly boost model performance. Updating the model with new reviews over time could help adaptation to evolving customer language and trends in sentiment.

Until improvements are made, the business should use the model’s predictions alongside human review to guide service improvements.

### Part V: Reporting

### I. Code Sources

D213 Webinars

D213 DataCamp courses

### J. Text Sources

GeeksforGeeks. (2023, June 5). *Understanding of LSTM networks*. https://www.geeksforgeeks.org/understanding-of-lstm-networks/

GeeksforGeeks. (2024, July 16). *What is tokenization?* https://www.geeksforgeeks.org/what-is-tokenization/

Ibm. (2025, April 17). *What is a recurrent neural network (RNN)?*. IBM. https://www.ibm.com/think/topics/recurrent-neural-networks