D213 Task 2 Sentiment Analysis Using Neural Networks

Scott Babcock WGU - MS, Data Analytics

Created: June 17 2025 *Last Edited: September 10 2025*

R Packages Used in Analysis

```
library(readr) # loading text files
library(dplyr) # data manipulation
library(stringr) # text manipulation
library(textclean) # text cleaning
library(textstem) # lemmatization
library(tm) # text pre-processing and cleaning
library(reticulate) # enabling python for tensorflow, error tracing
    use_condaenv("r-tf", required = TRUE) # establishing environment for tensorflow
library(tensorflow) # neural network
library(keras) # neural network
library(caret) # confusion matrix
```

A1, Research Question

Can the sentiment of reviews be predicted by using a Neural Network model?

A2, Goals

The analysis aims to feel confident in making predictions about whether a review is a positive or negative sentiment. The company could use what it has learned from correctly classifying reviews to determine how customers feel about products or the company in general. To achieve this goal, the data must be adequately cleaned and pre-processed, and a neural network model must be fit.

A3, Neural Network Identification

A Recurrent Neural Network (RNN) was used in the analysis, more specifically, an RNN variant called Long Short Term Memory (LSTM). LSTM is a good choice for sentiment analysis because it captures context from sequential data and retains long-term patterns (Sentiment Analysis, 2025).

B1, EDA

```
# load data and combine files
data_amazon <- read.delim("source_files/amazon_cells_labelled.txt", header = FALSE, sep = "\t", quote =
data_imdb <- read.delim("source_files/imdb_labelled.txt", header = FALSE, sep = "\t", quote = "", col.na
data_yelp <- read.delim("source_files/yelp_labelled.txt", header = FALSE, sep = "\t", quote = "", col.na

data_all <- bind_rows(data_amazon, data_imdb, data_yelp)

# explore data
cat("Summary of Combined Data Frame:\n")
```

```
## Summary of Combined Data Frame:
```

```
summary(data_all)
```

```
##      text               label       
##  Length:3000        Min.   :0.0  
##  Class :character   1st Qu.:0.0  
##  Mode  :character   Median :0.5  
##                     Mean   :0.5  
##                     3rd Qu.:1.0  
##                     Max.   :1.0
```

```
cat("Data Frame Dimensions:\n",dim(data_all),"\n")
```

```
## Data Frame Dimensions:
##  3000 2
```

```
cat("Table Summary of Sentiment:\n")
```

```
## Table Summary of Sentiment:
```

```
table(data_all$label)
```

```
## 
##    0    1 
## 1500 1500
```

```
cat("Number of NA in Data Frame:\n",sum(is.na(data_all)),"\n")
```

```
## Number of NA in Data Frame:
##  0
```

```
cat("Number of NULL in Data Frame:\n",sum(is.null(data_all)),"\n")
```

```
## Number of NULL in Data Frame:
##  0
```

```
# check for unusual characters
unusual_pre <- grepl("[^a-zA-Z0-9]",data_all$text)
unusual_length_pre <- length(unusual_pre)
cat("Total Number of Unusual Characters Instances:\n",unusual_length_pre,"\n")
```

```
## Total Number of Unusual Characters Instances:
##  3000
```

```
unusual_list_pre <- unlist(str_extract_all(data_all$text, "[^a-zA-Z0-9]"))
unusual_unique_pre <- unique(unusual_list_pre)
unusual_unique_length_pre <- length(unusual_unique_pre)
cat("Total Unique Unusual Characters Found:\n",unusual_unique_length_pre,"\n")
```

```
## Total Unique Unusual Characters Found:
##  27
```

```
cat("Unique Unusual Characters:\n",unusual_unique_pre,"\n")
```

```
## Unique Unusual Characters:
##    . , ! + " ' / ? - : ) ( & $ * ; % # [ ]  é  å  ê
```

```
# clean text - remove custom stop words and lemmatize
data_clean <- data_all

custom_stop_words  <- data.frame(word = c(
  "a", "an", "the", "and", "or", "but", "so", "in", "on", "at", "by", "with", "from", "to", "of", "for"
  "i", "you", "he", "she", "it", "we", "they", "me", "him", "her", "us", "them", "is", "are", "was", "w
  "be", "been", "being", "do", "does", "did", "have", "has", "had", "this", "that", "these", "those", "
  "here", "then", "when", "where", "why", "how", "my", "as","its","if"))

data_clean$text <- data_clean$text %>%
  str_to_lower() %>%
  replace_contraction() %>%
  replace_number() %>%
  replace_ordinal() %>%
  replace_symbol() %>%
  removeWords(custom_stop_words$word) %>%
  lemmatize_strings() %>%
  str_replace_all("[^a-z\\s]","") %>%
  str_squish()

# check for unusual characters post-cleaning
unusual <- grepl("[^a-zA-Z0-9]",data_clean$text)
unusual_length <- length(unusual)
cat("Total Number of Unusual Characters Instances:\n",unusual_length,"\n")
```

```
## Total Number of Unusual Characters Instances:
##  3000
```

```
unusual_list <- unlist(str_extract_all(data_clean$text, "[^a-zA-Z0-9]"))
unusual_unique <- unique(unusual_list)
unusual_unique_length <- length(unusual_unique)
cat("Total Unique Unusual Characters Found:\n",unusual_unique_length,"\n")
```

```
## Total Unique Unusual Characters Found:
##  1
```

```
cat("Unique Unusual Characters:\n",unusual_unique,"\n")
```

```
## Unique Unusual Characters:
##
```

```
# determine vocabulary size
tokenizer <- text_tokenizer()
tokenizer %>% fit_text_tokenizer(data_clean$text)

vocab_size <- length(tokenizer$word_index)
cat("Vocabulary Size:\n",vocab_size,"\n")
```

```
## Vocabulary Size:
##  4135
```

```
# determine max sequence length
seq_lengths <- data_clean$text %>%
  str_count("\\S+")
cat("Summary of Sequence Lengths:\n")
```

```
## Summary of Sequence Lengths:
```

```
summary(seq_lengths)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   4.000   6.000   7.194   9.000  48.000
```

```
max_seq_length <- max(seq_lengths)
mean_seq_length <- mean(seq_lengths)
cat("Max Sequence Length:\n",max_seq_length,"\n")
```

```
## Max Sequence Length:
##  48
```

```
cat("Mean Sequence Length:\n",mean_seq_length,"\n")
```

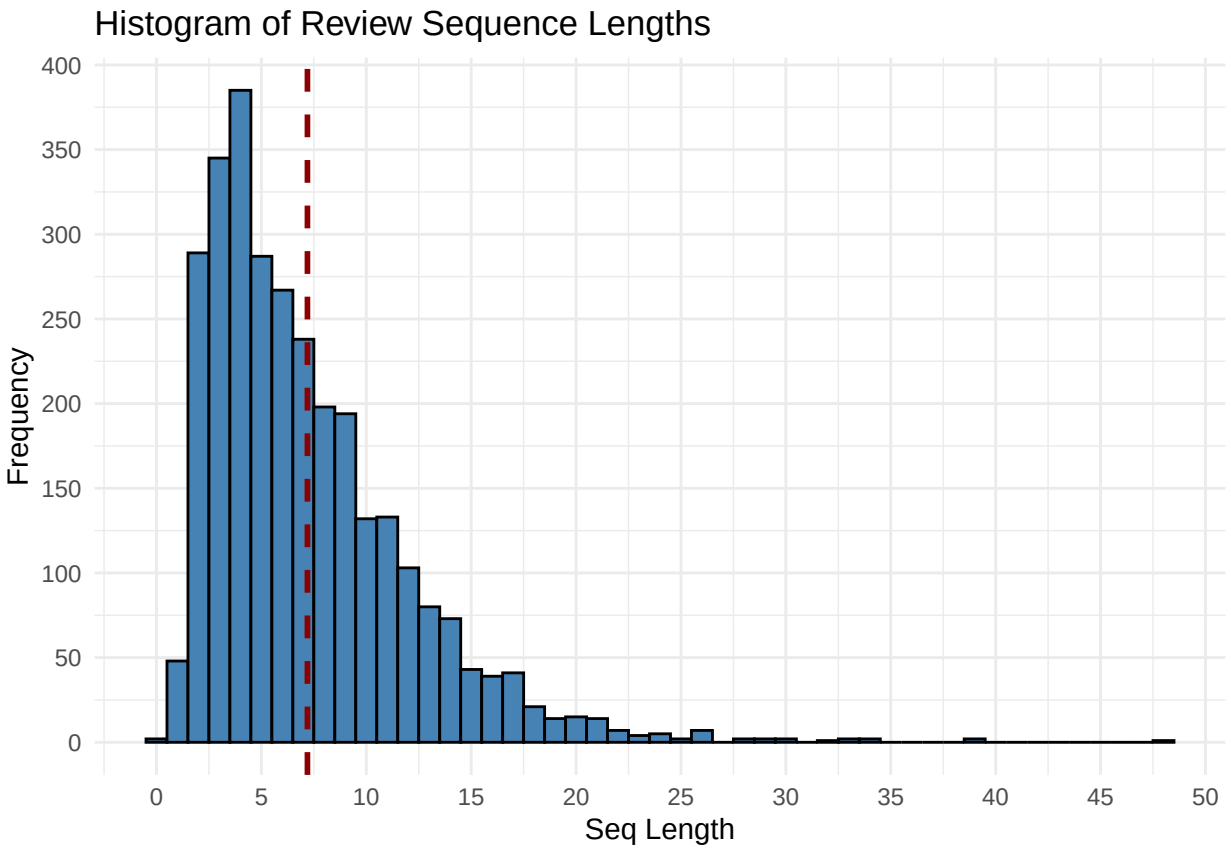
```
## Mean Sequence Length:
##  7.194333
```

```
seq_pctl_95 <- quantile(seq_lengths, 0.95) %>%
  ceiling()
cat("95th Percentile of Sequence Length:\n", seq_pctl_95,"\n")
```

```
## 95th Percentile of Sequence Length:
##  16
```

```
length_df <- data.frame(length = seq_lengths)

ggplot(length_df, aes(x = length))+
  geom_histogram(binwidth = 1, fill="steelblue",color = "black")+
  geom_vline(aes(xintercept = mean(length)), color = "darkred",linetype = "dashed",size = 1, show.legend
  scale_x_continuous(breaks = seq(0,50,5))+
  scale_y_continuous(breaks = seq(0,400,50))+
  labs(title = "Histogram of Review Sequence Lengths",
       x = "Seq Length",
       y = "Frequency")+
  theme_minimal()
```




```
maxlen <- as.integer(20)
```

Unusual Characters The data was checked for unusual characters that would impact model building. 3000 unusual character instances were identified, made up of 27 unique unusual characters.

Vocabulary Size After a cleaning process that involved removing a custom, smaller list of stop words and lemmatization, a vocabulary size of 4135 remained. The smaller list of stop words removed words with no value while retaining as much sentence structure as possible. Lemmatization was used to take words to their root form. The lemmatization helps reduce the vocabulary size but helps maintain the same sentiment.

Word Embedding Length The word embedding length, or the `output_dim` in Keras, is the size of the vector space in which the words will be embedded. It is the size of the output vectors for each word. The value is typically between 100 and 300 for deep learning models. Embedding lengths in this range are found to balance representational power and computational efficiency (Brownlee, 2021). A value of 100 was used in the analysis.

Max Sequence Length After cleaning, the maximum sequence length, or sentence length, is 48 words. A common approach is to use this maximum value in the modeling. Another approach uses the mean sentence length, which is 7.1943333. A value in the middle was chosen after viewing the distribution of review lengths. The distribution is right-skewed, with few occurrences beyond 20 words. Thus, 20 was selected as the sequence length, meaning reviews with less than 20 words will be padded with zeros to make the lengths uniform.

B2, Goals of Tokenization

The tokenization process aims to parse each sentence into individual words. The data should be thoroughly cleaned before or as part of the tokenization. Tensorflow requires uniform sentence lengths, so the text

must be padded with zeros for the sentences that do not reach the specified length, which in this analysis was 20.

```
# Determine vocab size for model input_dim and final tokenization/sequencing
vocab_input <- as.integer(vocab_size+1)

tokenizer <- text_tokenizer(num_words = vocab_input, oov_token = "OOV")
tokenizer %>% fit_text_tokenizer(data_clean$text)

sequences <- texts_to_sequences(tokenizer, data_clean$text)
word_index <- tokenizer$word_index
```

B3, Padding Process

The word embedding length was set at 20, as previously mentioned. This input means the sequencing process will create a vector with a width of 20. Reviews that have a length of less than 20 will be padded with zeros to stay uniform. In the analysis, padding was applied at the end of the strings because it produces a more logical presentation. An example of a padded sequence can be found below.

```
# Pad data based on maxlen
x_data <- pad_sequences(sequences, maxlen = maxlen, padding = "post")

cat("Dimensions of x_data:\n",dim(x_data),"\n")
```

```
## Dimensions of x_data:
##  3000 20
```

```
cat("Example of a Padded Sequence:\n",x_data[1, ],"\n")
```

```
## Example of a Padded Sequence:
##  33 74 248 502 17 1876 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
cat("Length of Padded Sequence:\n",length(x_data[1,]),"\n")
```

```
## Length of Padded Sequence:
##  20
```

B4, Categories of Sentiment

There are two categories of sentiment in the analysis: positive and negative. Positive sentiment was represented with a value of one, while negative sentiment was represented with a zero value. Sentiment is distributed equally, with 1,500 positive and negative reviews each. A `sigmoid` activation function was used in the output layer. The `sigmoid` function produces a zero-to-one output, which makes it ideal for a binary classification problem (Saeed, 2021).

```
cat("Breakdown of Sentiment:\n")
```

```
## Breakdown of Sentiment:
```

```r
table(data_clean$label)
```

```
##
##    0    1
## 1500 1500
```

B5, Preparation Steps

Preparation steps have been documented in-depth throughout the report. Below is a summary recap of the preparation and cleaning leading up to the train/test splits.

- All three files were loaded and combined into one data frame.
- The dimensions of the data frame (3000, 2) were identified, and the data was checked for any `NA` or `NULL` values.
- Exploratory analysis was performed, which included identifying the distribution of the sentiment labels and checking for unusual characters.
- A custom list of stop words was created, which included 56 words. The custom list was made to remove the obvious non-value-added words and retain as much sentiment as possible.
- Text cleaning was performed to replace contractions, numbers, ordinals, and symbols with their full-text values, remove the custom stop words, lemmatize the words (take them to their root form), convert all characters to lowercase, and remove punctuation and symbols.
- The text cleaning function was applied to the text as a new data frame.
- Additional EDA was performed to view the updated vocabulary size and review (sequence) lengths. A histogram of the review lengths helped determine the value used for maximum sequence length.
- The data was tokenized and padded to be fed into the RNN model.

The work above ensured the data was ready to be split into train and test sets. A train/test split size of 85/15 was used. An additional 15% would be partitioned off as a validation set within the training data. The training set would have 2,550 rows in total (3000 * 0.85), and within that, 2,167 rows (2550 * 0.85) would be used for training, and 383 rows (2550 * 0.15) would be used as an unseen validation set. The testing set will contain 450 rows (3000 * 0.15).

```r
# set seed for reproducibility and create splits
set.seed(444)
tensorflow::tf$random$set_seed(444)

indices <- sample(1:nrow(x_data))
y_data <- as.numeric(data_clean$label)

# Calculate split index as an integer
split_index <- as.integer(round(0.85 * length(indices)))

# Create train and test indices
train_indices <- indices[1:split_index]
test_indices <- indices[(split_index + 1):length(indices)]

x_train <- x_data[train_indices, ]
x_train <- matrix(as.integer(x_train),nrow=nrow(x_train),ncol=ncol(x_train))

x_test <- x_data[test_indices, ]
x_test <- matrix(as.integer(x_test),nrow=nrow(x_test),ncol=ncol(x_test))
```

```
y_train <- y_data[train_indices]
y_test <- y_data[test_indices]

y_train <- as.numeric(y_train)
y_test <- as.numeric(y_test)
y_train <- matrix(as.integer(y_train),ncol=1)
y_test <- matrix(as.integer(y_test),ncol=1)
```

B6, Prepared Data Set

A copy of the prepared data frame, and x and y training/test sets were included in the submission.

```
# write data to csv
write.csv(data_clean,
          "d213_task2_babcock_prepared_df.csv",
          row.names = FALSE)
write.csv(x_train,
          "d213_task2_babcock_prepared_x_train.csv",
          row.names = FALSE)
write.csv(x_test,
          "d213_task2_babcock_prepared_x_test.csv",
          row.names = FALSE)
write.csv(y_train,
          "d213_task2_babcock_prepared_y_train.csv",
          row.names = FALSE)
write.csv(y_test,
          "d213_task2_babcock_prepared_y_test.csv",
          row.names = FALSE)
```

C1, Model Summary

The model was built using the Keras Sequential API, which runs on a Tensorflow backend engine. The `model$summary()` output can be found below. The summary output provides information on the layers included in the model and their order, the output shape of each layer, and the number of parameters, or weights, in each layer (Brownlee, 2019).

```
# build model
model <- keras_model_sequential()
  model$add(layer_embedding(input_dim = vocab_input, output_dim = 100, input_length = maxlen))
  model$add(bidirectional(layer= layer_lstm(units = 16, return_sequences = TRUE, dropout = 0.2, recurrer
  model$add(bidirectional(layer= layer_lstm(units = 16, return_sequences = TRUE, dropout = 0.2, recurrer
  model$add(bidirectional(layer= layer_lstm(units = 16, dropout = 0.2, recurrent_dropout = 0.2)))
  model$add(layer_dense(units = 1, activation = "sigmoid"))

model$compile(
  loss = "binary_crossentropy",
  optimizer = 'adam',
  metrics = list("accuracy"))

# Train Model
```

```r
callback <- list(
  callback_early_stopping(monitor = "val_loss", patience = 3, restore_best_weights = TRUE),
  callback_reduce_lr_on_plateau(monitor = "val_loss", factor = 0.5, patience = 2))

history <- model$fit(
  x_train,
  y_train,
  epochs = as.integer(25),
  batch_size = as.integer(32),
  validation_split = 0.15,
  callbacks = list(callback),
  verbose = 0
)

model$summary()
```

```
## Model: "sequential"
##
##  Layer (type)                        Output Shape                    Param #
##
##  embedding (Embedding)               (None, 20, 100)                 413,600
##
##  bidirectional (Bidirectional)       (None, 20, 32)                   14,976
##
##  bidirectional_1 (Bidirectional)     (None, 20, 32)                    6,272
##
##  bidirectional_2 (Bidirectional)     (None, 32)                        6,272
##
##  dense (Dense)                       (None, 1)                            33
##
##  Total params: 1,323,461 (5.05 MB)
##  Trainable params: 441,153 (1.68 MB)
##  Non-trainable params: 0 (0.00 B)
##  Optimizer params: 882,308 (3.37 MB)
```

C2, Network Architecture

The final model has five layers: one embedding layer, three hidden layers, and one output layer. The embedding layer receives the sequence of integers from the tokenization and sequencing steps and creates a dense vector for each integer. The embedding layer was the largest layer with 413,600 parameters (calculated by the vocabulary size +1 multiplied by the output dimension 100). The three hidden layers are bidirectional Long Short-Term Memory (LSTM) layers. These layers allow the model to learn from prior and future data or backward and forward contexts (Brownlee, 2021). LSTM is especially useful in sentiment analysis, where the sequence of words is vital. Each of these layers had 16 nodes, which ended up being 32 in total because they were bidirectional (16 forward, 16 backward). The final layer is a dense layer that produces one binary classification output. The `sigmoid` activation function produces a value between 0 and 1, which can be interpreted as the probability of sentiment (Saeed, 2021). If the output of a given review is 0.36, that will get rounded down to zero, which signifies negative sentiment.

C3, Hyperparameters

Activation Functions The default activation function for the Keras LSTM layer is `tanh`, which was kept as-is in the analysis. The `sigmoid` activation function was used in the final output layer, as this is best for binary classification. It produces an output ranging from zero to one.

Nodes per Layer Each hidden layer has 16 nodes each, which is effectively 32 since they are bidirectional. These were determined through trial and error, observing which inputs produced a more accurate model.

Loss Function The loss function used in the `model$compile()` step was `binary cross-entropy`, which is suitable for a binary classification problem. It measures the difference between the predicted probability and the actual label. A higher penalty is applied when the model is confident in the prediction and incorrect (Allaire & Chollet, 2017).

Optimizer The `Adam` optimizer was used in the `model$compile()` step. The `Adam` optimizer is an excellent model choice due to its adaptive learning rate. It often requires less tuning.

Stopping Criteria An early stopping criterion was added to the `model$fit()` step. The stopping criteria was set to monitor validation loss with a patience of three. The training would stop if the model went through three epochs without improving. An additional component was added to reduce the learning rate by 0.5 if the validation loss did not improve for two consecutive epochs. The learning rate reduction can help the model get out of plateaus and give it a chance to improve before stopping.

Evaluation Metric Accuracy was the chosen model evaluation metric. It was chosen because it is an easy-to-interpret and effective metric.

D1, Stopping Criteria

The stopping criteria used in the model increase efficiency. The total number of epochs can be higher when early stopping is implemented because it will automatically stop training if there is no improvement for a specified number of epochs, in this case, three. An additional component was added to decrease the learning rate if there was no improvement for a specified number of epochs, in this case, two. The learning rate reduction allows the model to make improvements before stopping.

Below is an image showing the training stopping after 'r epoch_final' epochs when the model was set to 25. The model decreased the training rate in its final run and saw no improvement.

D2, Model Fit

```
# Extract metrics from the history object
acc <- history$history$accuracy
acc_final <- round(tail(acc,1),4)
val_acc <- history$history$val_accuracy
val_acc_final <- round(tail(val_acc,1),4)
loss <- history$history$loss
loss_final <- round(tail(loss,1),4)
val_loss <- history$history$val_loss
val_loss_final <- round(tail(val_loss,1),4)
epochs <- seq_along(acc)
epoch_final <- tail(epochs,1)

cat("Final Training Accuracy: ",acc_final,"\n")
```

```
## Final Training Accuracy:  0.9728
```

```
cat("Final Validation Accuracy: ",val_acc_final,"\n")
```

```
## Final Validation Accuracy:  0.7859
```

```
cat("Final Training Loss: ",loss_final,"\n")
```

```
## Final Training Loss:  0.1019
```

```
cat("Final Validation Loss: ",val_loss_final,"\n")
```

```
## Final Validation Loss:  0.7114
```

The model appears to be a good fit. It produces a training accuracy of 0.9728, which is stellar. The training loss was low at 0.1019. The model performed well in terms of accuracy on the validation set with 0.7859 accuracy. However, the validation loss was high at 0.7114, which indicates that the model is overfitting. Measures were taken to reduce overfitting by adding dropouts to layers and adding early stopping criteria. Dropout is when some neurons in a layer are deactivated, preventing overreliance on any single neuron (Dropout, n.d.).

```
# Evaluate Model with Test Data
results <- model$evaluate(x_test, y_test, verbose = 0)
test_acc <- round(results[2],4)
test_loss <- round(results[1],4)

cat("Test Accuracy: ",test_acc,"\n")
```

```
## Test Accuracy:  0.8444
```

```
cat("Test Loss: ",test_loss,"\n")
```

```
## Test Loss:  0.3966
```

```
# Generate Predictions
y_pred_prob <- model$predict(x_test, verbose = 0)
y_pred <- ifelse(y_pred_prob > 0.5, 1, 0)

# Create confusion matrix
confusion <- confusionMatrix(factor(y_pred), factor(y_test))
conf_sensitivity <- round(confusion$byClass["Sensitivity"],4)
conf_specificity <- round(confusion$byClass["Specificity"],4)

confusion
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 204  36
```

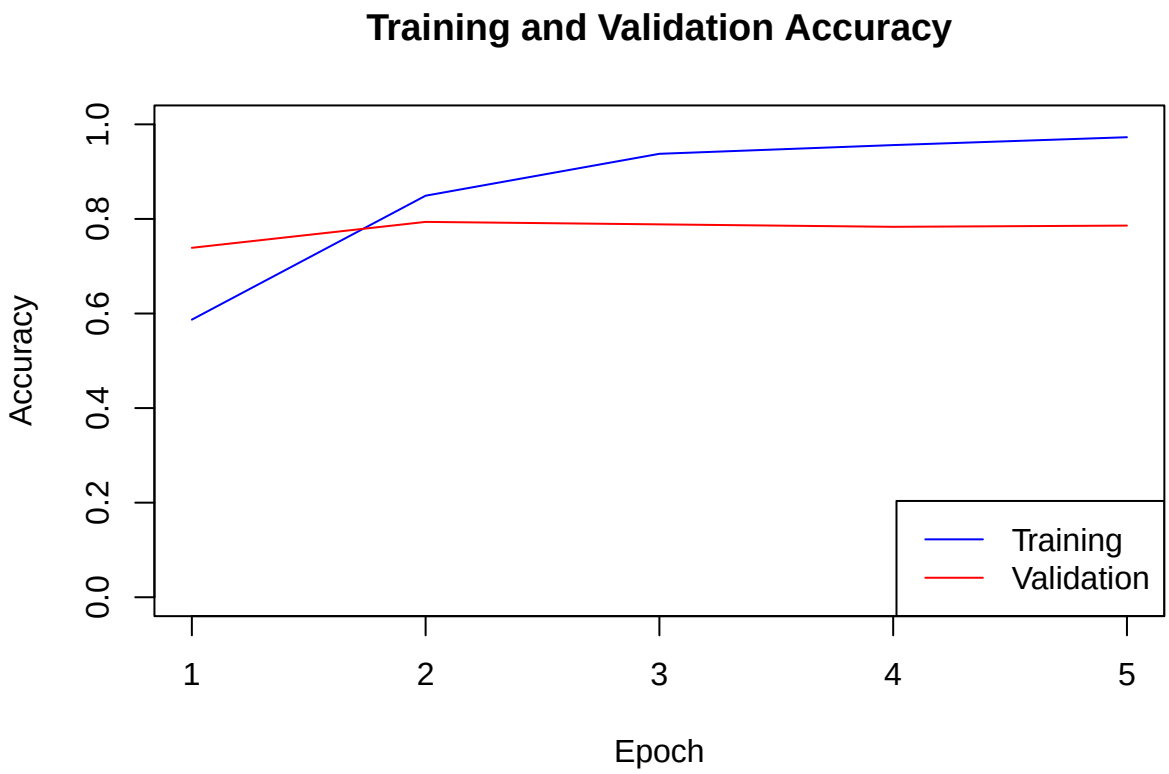
```
##          1  34 176
##
##                Accuracy : 0.8444
##                  95% CI : (0.8076, 0.8767)
##     No Information Rate : 0.5289
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6877
##
##  Mcnemar's Test P-Value : 0.9049
##
##             Sensitivity : 0.8571
##             Specificity : 0.8302
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 0.8381
##              Prevalence : 0.5289
##          Detection Rate : 0.4533
##    Detection Prevalence : 0.5333
##       Balanced Accuracy : 0.8437
##
##        'Positive' Class : 0
##
```

While the validation loss was concerning, the model performed admirably on the test set. The test accuracy was 0.8444, and the test loss was 0.3966. These are both indicators of a good model fit. When assessing the confusion matrix, the sensitivity was 0.8571, which indicates that the model is excellent at predicting positive sentiment. The specificity was 0.8302, which indicates that the model also does a fine job predicting negative sentiment, albeit not as well as positive sentiment.

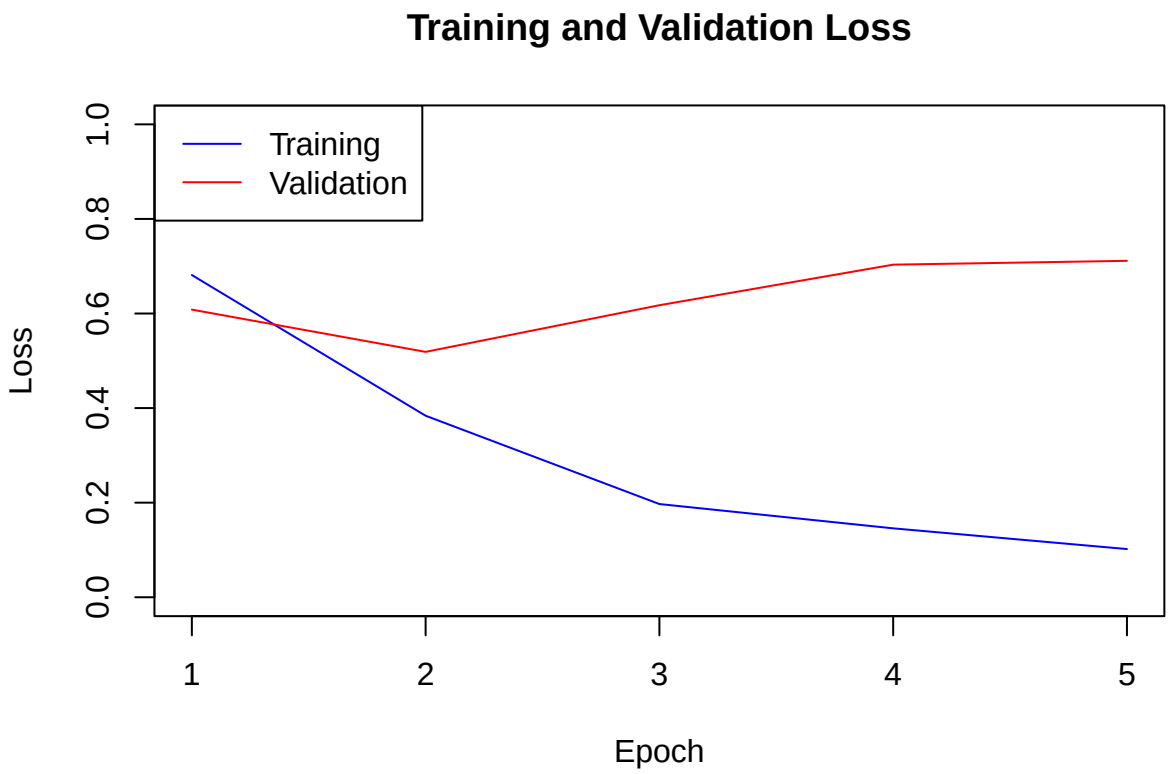
D3, Visualizations

Below are two visualizations tracking the accuracy and loss of training and validation sets across each epoch. The model was set to run for 25 epochs; however, due to the early stopping criteria, it concluded at 5 epochs after no improvement. The accuracy lines remain close. However, the loss shows a widening gap across epochs.

```
# Plot Training & Validation Accuracy
plot(epochs, acc, type = "l", col = "blue", ylim = c(0, 1),
     xlab = "Epoch", ylab = "Accuracy", main = "Training and Validation Accuracy")
lines(epochs, val_acc, col = "red")
legend("bottomright", legend = c("Training", "Validation"), col = c("blue", "red"), lty = 1)
```




```r
# Plot Training & Validation Loss
plot(epochs, loss, type = "l", col = "blue", ylim = c(0, 1),
     xlab = "Epoch", ylab = "Loss", main = "Training and Validation Loss")
lines(epochs, val_loss, col = "red")
legend("topleft", legend = c("Training", "Validation"), col = c("blue", "red"), lty = 1)
```



D4, Accuracy

The trained model is very accurate. The model produced training accuracy of 0.9728, validation accuracy of 0.7859, and then performed well on the unseen test data with an accuracy of 0.8444. The test set contained 450 rows, which is a decent sample size. It would be interesting to see if the accuracy held on a larger unseen data set.

E, Neural Network Code

The code below saves the model as a Keras model object. This is a newer format that is more flexible than the previous h5 format.

```
# Save model
model$save("text_sentiment_model.keras")
```

F, Functionality of Neural Network

The neural network functions by receiving a matrix of tokenized words, which get converted to numeric values. It analyzes words that produce positive or negative sentiments and then makes predictions on the training, validation, and test sets. The network was set up using bidirectional LSTM layers, which allows the network to learn from backward and forward contexts (Bidirectional LSTM, 2025). The output is a probability of sentiment ranging from zero to one. Values below 0.5 are considered negative sentiment,

and values above 0.5 are considered positive sentiment. This architecture produced good results, with the model producing high accuracy on the unseen data.

G, Recommended Course of Action

The model is production-ready based on the model's accuracy on unseen validation and test data. The company could use this model to analyze reviews of the company or company's products and determine overall sentiment. These insights are valuable because they can help drive change. If customers have a low opinion of the company, leadership can help drive positive changes and improve sentiment. It would be wise to continue to refine the model in parallel by adding more data and tuning model parameters. The model produced good results, but it was on a relatively small data set (3000 rows). Adding more data and still getting similar results would continue to increase confidence in the model.

Appendix

H, Reporting

This report was created in R Studio using an R Markdown format. It was knitted to HTML for submission.

I, Code Sources

Awan, A. (February 6, 2023). *Building Neural Network (NN) Models in R*. DataCamp. Retrieved June 22, 2025, from (https://www.datacamp.com/tutorial/neural-network-models-r)

Guide to Keras basics (n.d.). TensorFlow for R. Retrieved June 22, 2025, from (https://tensorflow.rstudio.com/guides/keras/basics).

Jones, K. (n.d.). *Introduction to natural language processing in R* [MOOC]. DataCamp. Retrieved June 22, 2025, from (https://app.datacamp.com/learn/courses/introduction-to-natural-language-processing-in-r)

Turner, D. (January 12, 2024). *Quick start guide to neural networks in R with Keras and Tensorflow*. Dusty S Turner. Retrieved June 23, 2025, from (https://dustysturner.com/post/2024-01-13-quick-start-guide-to-neural-networks-in-r-with-keras-and-tensorflow/)

Verma, A. (July 20, 2020). *Building a neural net from scratch using R – part 1*. R Views. Retrieved June 22, 2025, from (https://rviews.rstudio.com/2020/07/20/shallow-neural-net-from-scratch-using-r-part-1/)

Willems, K. (February 12, 2019). *Keras: deep learning in R*. DataCamp. Retrieved June 23, 2025, from (https://www.datacamp.com/tutorial/keras-r-deep-learning)

J, Content Sources

Allaire, J.J. and Chollet, F (December 6, 2017). *Deep learning for text classification with Keras*. Posit. Retrieved June 22, 2025, from (https://blogs.rstudio.com/ai/posts/2017-12-07-text-classification-with-keras/)

Bidirectional LSTM in NLP (May 28, 2025). Geeks for Geeks. Retrieved June 23, 2025, from (https://www.geeksforgeeks.org/nlp/bidirectional-lstm-in-nlp/)

Brownlee, J. (September 11, 2019). *How to visualize a deep learning neural network model in Keras*. Machine Learning Mastery. Retrieved June 25, 2025, from (https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/)

Brownlee, J. (February 2, 2021). *How to use word embedding layers for deep learning with Keras*. Machine Learning Mastery. Retrieved June 25, 2025, from (https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/)

Dropout: a comprehensive exploration (n.d.). Flyriver. Retrieved June 23, 2025, from (https://www.flyriver.com/g/dropout)

Saeed, M. (August 18, 2021). *A gentle introduction to sigmoid function*. Machine Learning Mastery. Retrieved June 23, 2025, from (https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/)

Sentiment analysis with an recurrent neural network (RNN) (May 27, 2025). Geeks for Geeks. Retrieved June 22, 2025, from (https://www.geeksforgeeks.org/python/sentiment-analysis-with-an-recurrent-neural-networks-rnn/)