D213: Advanced Data Analytics Task 2

Justin Huynh

Student ID: 012229514

M.S. Data Analytics

A1. Research Question

My research question for this project is "How can sentiment analysis of customer reviews from the Amazon, IMDb, and Yelp files be used to predict the likelihood of customer satisfaction with these products, movies, and services?"

A2. Objectives or Goals

For this analysis, we will have several objectives and goals that's defined below:

- Objective 1: Perform Sentiment Classification
  - Goal: Use neural network models to classify reviews as positive or negative based on the labeled dataset.
- Objective 2: Identify Key Patterns in Sentiment
  - Goal: Analyze the most common words, phrases, and topics associated with positive and negative reviews across the different platforms.
- Objective 3: Model Accuracy and Evaluation
  - Goal: Make sure the neural network model achieves a high level of accuracy when predicting sentiment in customer reviews.
- Objective 4: Provide Actionable Insights
  - Goal: Generate insights that can help organizations understand customer satisfaction that can improve the services, products, and customer engagement.

A3. Prescribed Network

A suitable type of neural network for this task is the Long Short-Term Memory (LSTM) network. LSTMs are a type of recurrent neural network (RNN) that are effective at learning patterns in sequential data like text sequences. They are well-suited for tasks that involve language processing and can be trained to predict the sentiment of text by learning word usage and context over time. This makes LSTMs ideal for performing text classification tasks on the sentiment-labeled sentences from the UCI dataset.

B1. Data Exploration

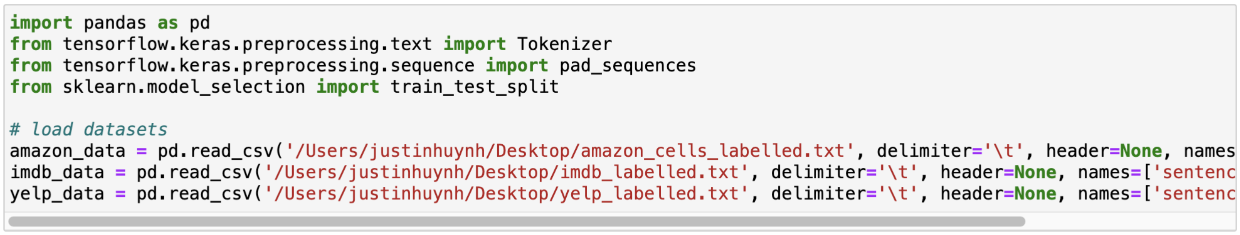
For the UCI dataset, this is the exploratory analysis for each of the following elements:

- Presence of Unusual Characters
    - To check for unusual characters like emojis or non-English characters, we can run a check across the dataset. Generally, reviews from platforms like Amazon, IMDb, and Yelp may contain punctuation, special symbols, or misspellings but not many non-English characters or emojis.
- Vocabulary Size
    - Vocabulary size is the total number of unique words in the dataset. We can count the number of distinct tokens (words) in the corpus after tokenization. The larger the vocabulary, the more robust the model will need to be to understand the diverse words in the text.
- Proposed Word Embedding Length
    - Word embedding length determines how each word is represented as a dense vector in the neural network. A commonly used embedding size is 100 or 300, which balances accuracy and cost. We can experiment with different embedding sizes (like 100 or 300 dimensions) based on how large the vocabulary is and the complexity of the model.
- Statistical Justification for the Chosen Maximum Sequence Length

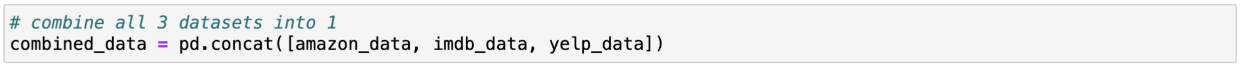
- ○ We'll calculate the length of the text sequences and choose an optimal maximum sequence length. This could be determined by setting the length to cover 90-95% of the text samples, ensuring that most sequences are of a reasonable size without truncating too many words. For example, we might find that a maximum sequence length of 50 words is sufficient for most reviews.

B2. Tokenization

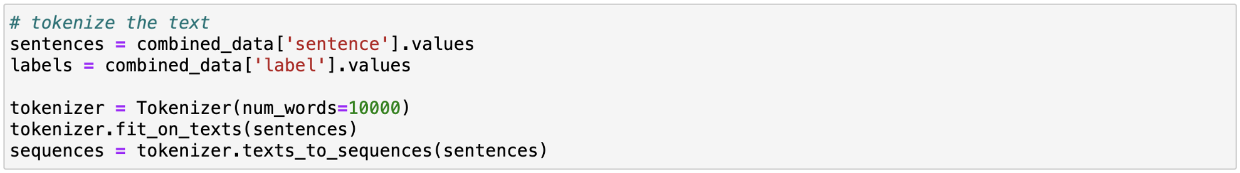
The goal of the tokenization process is to split the text into individual words or tokens and convert them into numerical format that the neural network can understand.



```
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

# load datasets
amazon_data = pd.read_csv('/Users/justinhuynh/Desktop/amazon_cells_labelled.txt', delimiter='\t', header=None, names
imdb_data = pd.read_csv('/Users/justinhuynh/Desktop/imdb_labelled.txt', delimiter='\t', header=None, names=['sentenc
yelp_data = pd.read_csv('/Users/justinhuynh/Desktop/yelp_labelled.txt', delimiter='\t', header=None, names=['sentenc
```



```
# combine all 3 datasets into 1
combined_data = pd.concat([amazon_data, imdb_data, yelp_data])
```

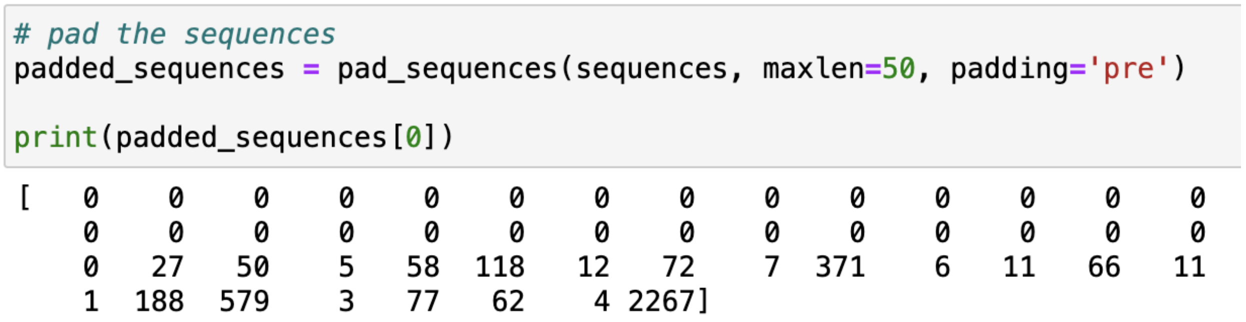


```
# tokenize the text
sentences = combined_data['sentence'].values
labels = combined_data['label'].values

tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(sentences)
sequences = tokenizer.texts_to_sequences(sentences)
```

See code attached in WGU_D213_Task_2.ipynb.

B3. Padding Process

Padding is used to standardize the length of all input sequences when working with neural networks to ensure uniformity. Padding adds extra elements, usually zeros, to the sequences to make them of equal length. These padded sequences are then fed into the neural network. The padding can either occur before the sequence (pre-padding where zeros will be added in the beginning of the sequence) or after the sequence (post-padding where zeros will be added to the end of the sequence). We will be using pre-padding since it’s more common in NLP tasks because the most relevant information appears at the end of the sequence, and we want to preserve that part of the sequence by padding at the front.



```
# pad the sequences
padded_sequences = pad_sequences(sequences, maxlen=50, padding='pre')

print(padded_sequences[0])
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0   27   50    5   58  118   12   72    7  371    6   11   66   11
    1  188  579    3   77   62    4 2267]
```

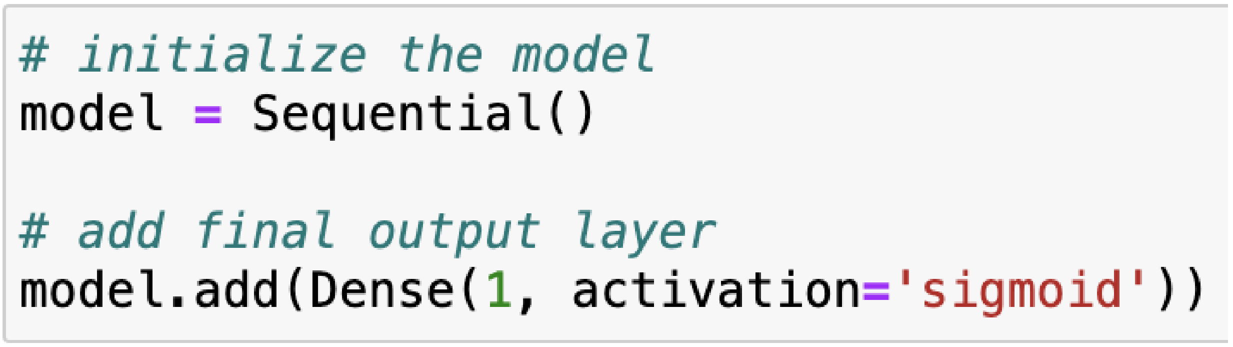
See code attached in WGU_D213_Task_2.ipynb.

B4. Categories of Sentiment

This is a binary classification task with two categories of sentiment:

- Positive Sentiment (1)
- Negative Sentiment (0)

Since this is a binary classification problem, the appropriate activation function for the final dense layer is the sigmoid function. This function outputs a probability between 0 and 1, which corresponds to the two sentiment classes.



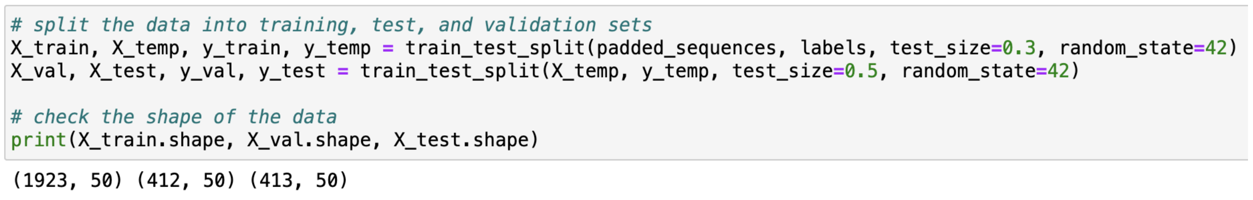
```
# initialize the model
model = Sequential()

# add final output layer
model.add(Dense(1, activation='sigmoid'))
```

See code attached in WGU_D213_Task_2.ipynb.

B5: Steps to Prepare the Data

These were the following steps to prepare the data for analysis:

- Data Loading
    - The three datasets (Amazon, IMDb, and Yelp) were loaded into a combined dataset containing sentences (reviews) and labels (0 for negative sentiment, 1 for positive sentiment).
- Tokenization
    - We applied the Tokenizer from TensorFlow, which converts text into sequences of integers. The tokenizer is configured to keep only the top 10,000 most frequent words in the vocabulary.
- Padding
    - Sequences were padded to a maximum length of 50 tokens. Pre-padding was used to ensure that all sequences are of uniform length.
- Splitting the Data
    - The dataset was split into training, validation, and test sets using an industry-standard ratio:
        - Training Set (70%): Used to train the model.
        - Validation Set (15%): Used to tune model parameters during training.
        - Test Set (15%): Used to evaluate the model’s performance on unseen data after training.



```
# split the data into training, test, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(padded_sequences, labels, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# check the shape of the data
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(1923, 50) (412, 50) (413, 50)
```

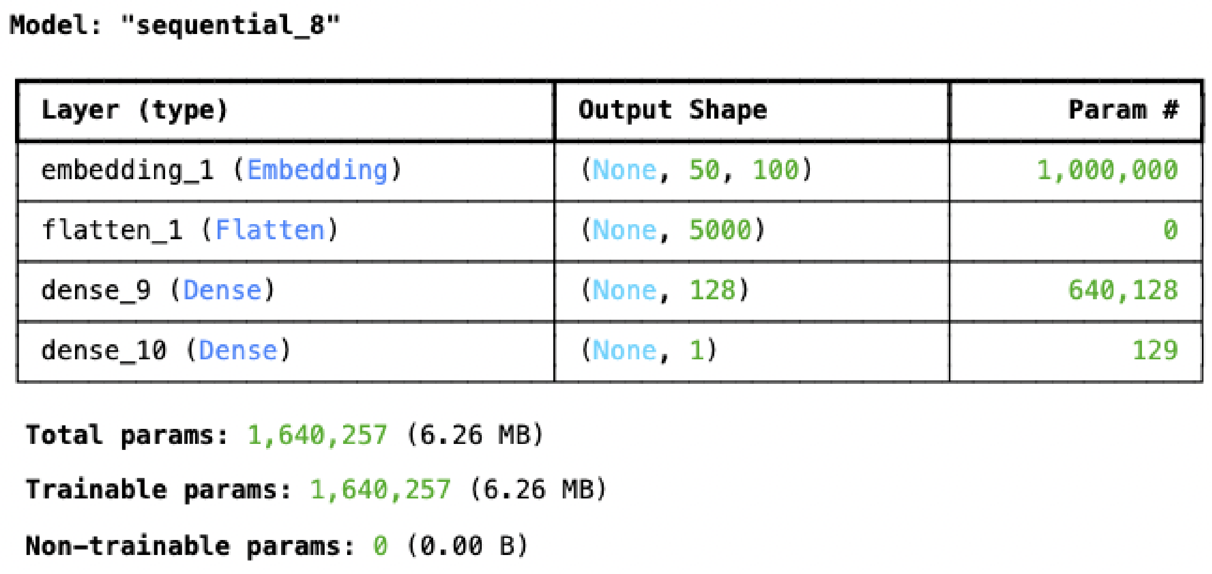
The result of the distribution showed 1,923 samples for the training set, 412 samples for the validation set, and 413 samples for the test set.

See code attached in WGU_D213_Task_2.ipynb.

B6: Prepared Data Set

A copy of the fully prepared data set will be submitted as 'prepared_sentiment_data_d213_task2.csv'.

C1: Model Summary



```
Model: "sequential_8"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                   ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)        │ (None, 50, 100)        │     1,000,000 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)            │ (None, 5000)           │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_9 (Dense)                │ (None, 128)            │       640,128 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_10 (Dense)               │ (None, 1)              │           129 │
└────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,640,257 (6.26 MB)
 Trainable params: 1,640,257 (6.26 MB)
 Non-trainable params: 0 (0.00 B)
```

See code attached in WGU_D213_Task_2.ipynb.

C2: Network Architecture

There are 4 total layers with the type of layer each described below:

- Embedding Layer

    - Converts words into dense vector representations.
- Flatten Layer
    - Flattens the 3D tensor (50, 100) into a 1D array of 5,000 elements.
- Dense Layer
    - A hidden dense layer with 128 units and ReLU activation.
- Output Layer
    - A single neuron with a sigmoid activation for binary classification.

There are 1,640,257 trainable parameters as shown in the model summary.

- The Embedding Layer has 1,000,000 parameters. This is calculated as 10,000 words * 100 embedding dimensions.
- The Dense Layer has 640,128 parameters calculated as 5000 (from Flatten layer) * 128 (dense units) + 128 (bias terms).
- The Output Layer has 129 parameters calculated as 128 (dense units) * 1 (output unit) + 1 (bias term).

C3: Hyperparameters

This is the justification of the choice of hyperparameters in the following 6 elements:

- Activation Functions
    - ReLU (Rectified Linear Unit) in the hidden Dense layer allows the model to capture complex relationships without introducing non-linearities that could cause vanishing gradients.
    - Sigmoid in the output layer compresses the output to a range between 0 and 1, which is ideal for binary classification because it outputs the probability of the positive class.

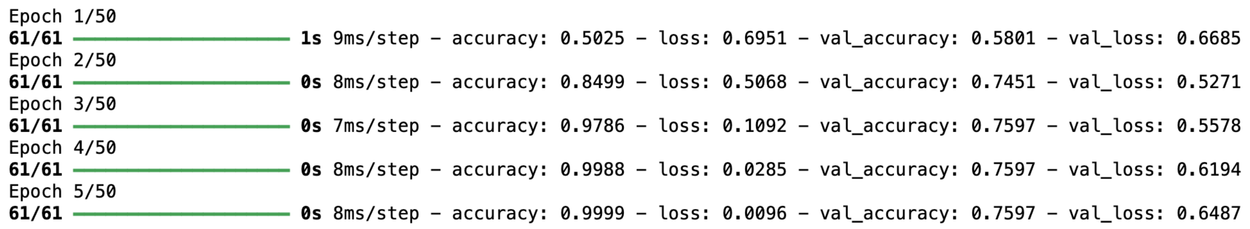
- Number of Nodes per Layer
    - 128 nodes were chosen for the hidden Dense layer to provide sufficient capacity for learning without overfitting. This is a commonly used size for medium-sized datasets and can be adjusted through experimentation.
- Loss Function
    - Binary Cross Entropy is the most suitable loss function for binary classification since it measures the difference between the predicted probability and the actual class (0 or 1).
- Optimizer
    - Adam Optimizer combines the benefits of AdaGrad and RMSProp, which provides adaptive learning rates and faster convergence. It is generally recommended for most deep learning tasks, especially when working with textual data.
- Stopping Criteria
    - Early stopping can be used to monitor the validation loss and prevent the model from overfitting by stopping training when the validation performance stops improving. This avoids unnecessary training and reduces the risk of overfitting.
- Evaluation Metric
    - Since this is a balanced binary classification problem, accuracy is an appropriate evaluation metric. It measures how often the model's predictions are correct.

D1: Stopping Criteria

In this model training process, early stopping was used to control the number of training epochs and prevent overfitting. Early stopping works by monitoring the validation loss and halting training when no further improvement is seen after a set number of epochs. This ensures that the model does not overfit the training data while still learning effectively.

The model was initially set to train for a maximum of 50 epochs, which allows enough time for the model to learn and converge. The early stopping callback was configured with a patience of 3 epochs, meaning that if the validation loss does not improve for 3 consecutive epochs, the training will be stopped automatically. In the output provided, we can see that the model was stopped early, after 5 epochs, due to early stopping. Although the initial number of epochs was set to 50, the early stopping mechanism recognized that the model was no longer improving on the validation loss after the third epoch.

The impact of early stopping ensured that the model did not continue training past the point where it was beneficial. This helped prevent overfitting to the training data and kept the model's performance more generalized to unseen data. The validation accuracy plateaued at 75.97% after epoch 3 while the validation loss began to increase slightly, signaling that further training would likely lead to overfitting. Early stopping correctly intervened to stop training before the model could overfit.



```
Epoch 1/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5025 - loss: 0.6951 - val_accuracy: 0.5801 - val_loss: 0.6685
Epoch 2/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8499 - loss: 0.5068 - val_accuracy: 0.7451 - val_loss: 0.5271
Epoch 3/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9786 - loss: 0.1092 - val_accuracy: 0.7597 - val_loss: 0.5578
Epoch 4/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9988 - loss: 0.0285 - val_accuracy: 0.7597 - val_loss: 0.6194
Epoch 5/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9999 - loss: 0.0096 - val_accuracy: 0.7597 - val_loss: 0.6487
```

See code attached in WGU_D213_Task_2.ipynb.

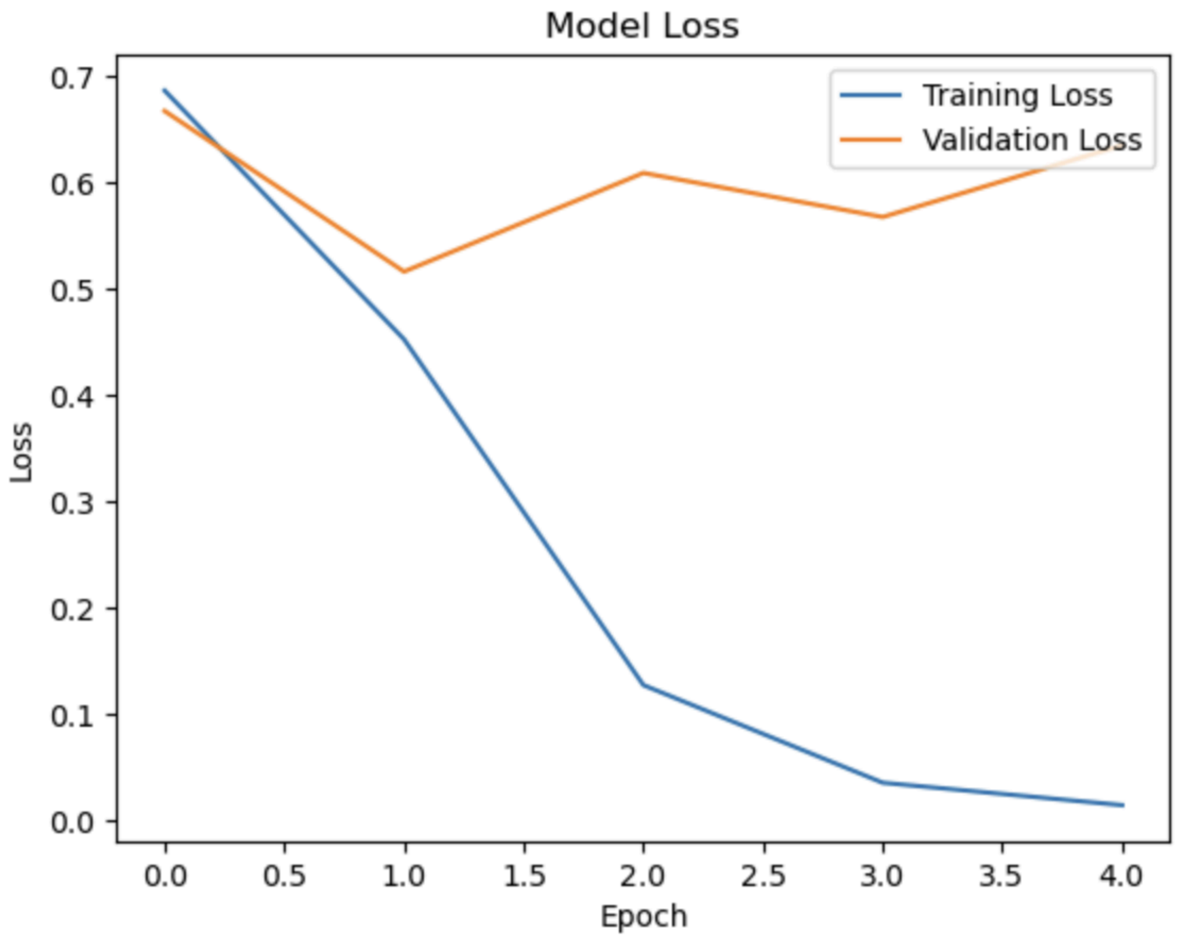
D2: Fitness

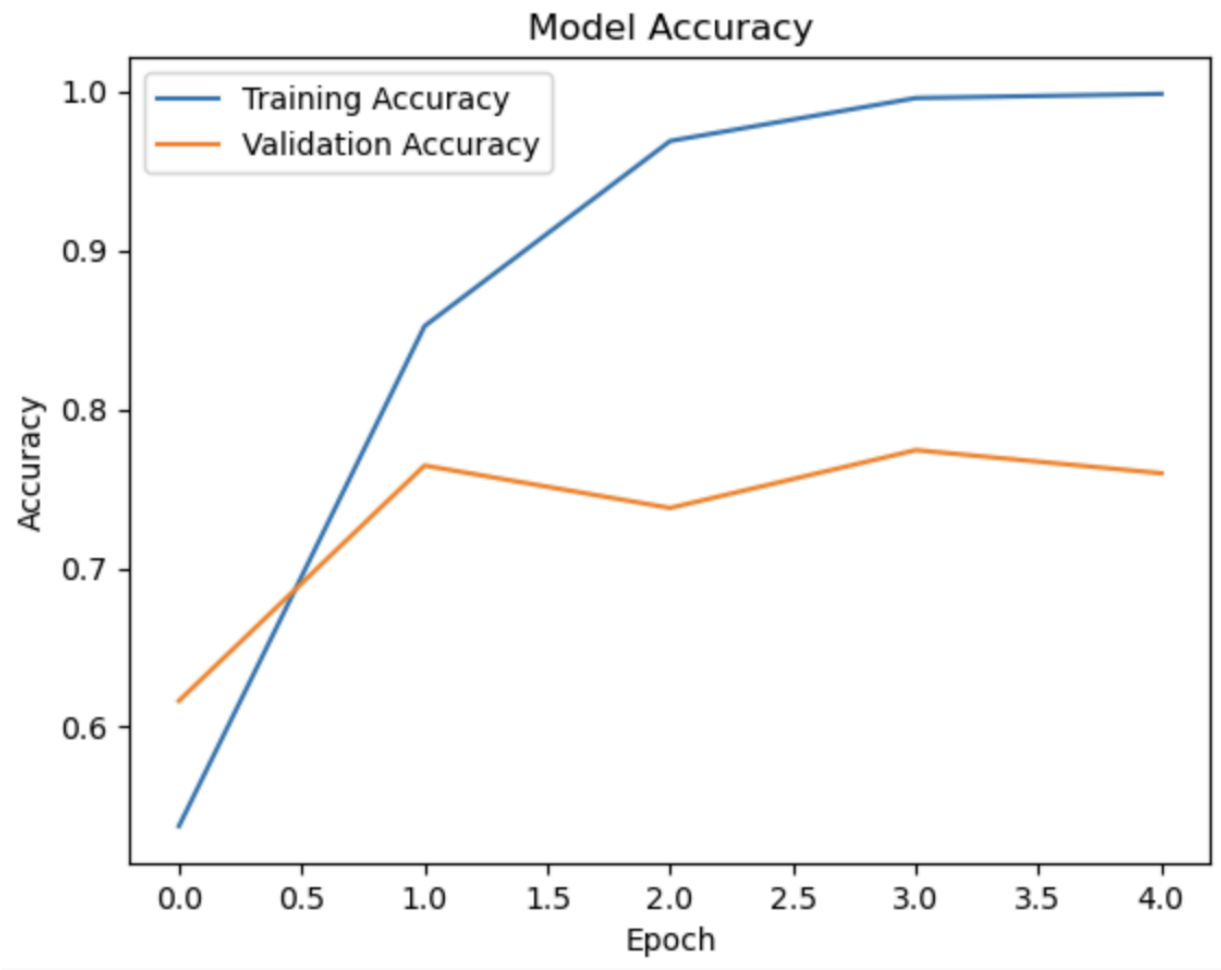
The model's fitness is measured by its performance on both the training and validation sets. The key indicators of fitness include:

- Training Loss and Accuracy
    - These reflect how well the model fits the training data.
- Validation Loss and Accuracy
    - These show how well the model generalizes to unseen data.

Overfitting can occur when the model performs well on the training data but poorly on the validation data. This typically results in the validation loss increasing while the training loss decreases. In this case, early stopping was used to stop training before overfitting could occur, ensuring that the model generalizes better.

D3: Training Process

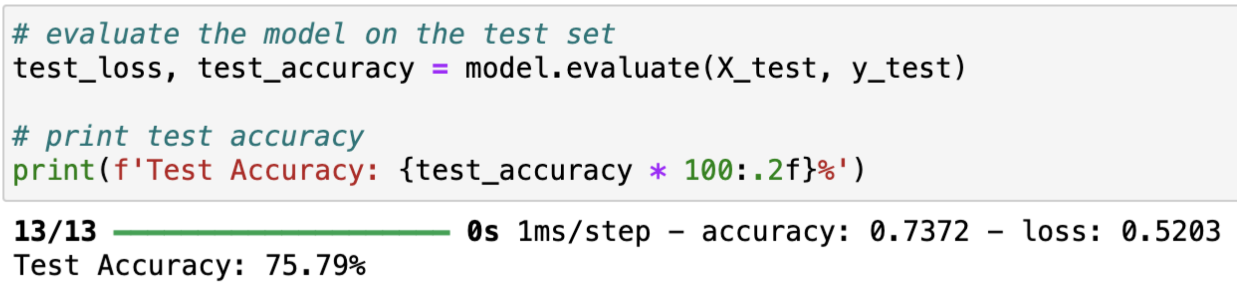

Model Loss
Training Loss
Validation Loss
Loss
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



See code attached in WGU_D213_Task_2.ipynb.

D4: Predictive Accuracy

Based on the test accuracy evaluation, the model achieved 75.79% accuracy on the test set, indicating that it correctly classified approximately 3 out of 4 reviews in the test set. The test loss was 0.5203, which shows that the model has a moderate level of error when predicting the sentiment for new data points. The model performs consistently across the training, validation, and test sets since the accuracy does not drop significantly between these datasets. This is a sign that the model is not overfitting and generalization to new data is strong.



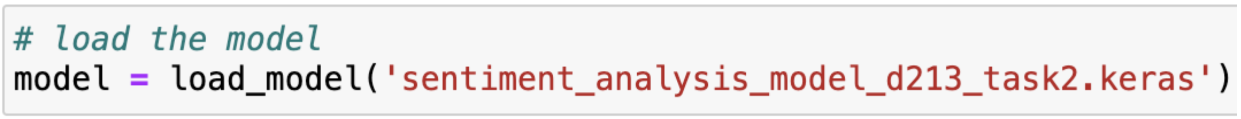
```
# evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

# print test accuracy
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7372 - loss: 0.5203
Test Accuracy: 75.79%
```

See code attached in WGU_D213_Task_2.ipynb.

E: Code

A copy of the code will be submitted as ‘sentiment_analysis_model_d213_task2.keras’. Since it is a binary file and not a text file, it should be loaded programmatically with Keras using the code below.



```
# load the model
model = load_model('sentiment_analysis_model_d213_task2.keras')
```

See code attached in WGU_D213_Task_2.ipynb.

F: Functionality

The neural network built in this research was designed to classify the sentiment of customer reviews as either positive or negative, based on a dataset of reviews from Amazon, IMDb, and Yelp. Here’s how the architecture and its components contributed to its functionality:

- Embedding Layer
  - The embedding layer was the first step in the architecture, which converted the input words into dense, continuous vector representations. This allowed the model to capture word-level relationships and context within the reviews. The embedding dimension of 100 was chosen as a

balance between learning meaningful relationships and efficiency for computation.

- Dense Layers
    - After flattening the output from the embedding layer, we added a fully connected dense layer with 128 nodes and a ReLU activation function. This layer helped the model learn complex relationships between the input features (words) and the target labels (sentiment).
    - The final output layer used a sigmoid activation function to output a probability, indicating whether the sentiment was positive or negative. This was appropriate for the binary classification task.
- Early Stopping
    - Early stopping was implemented to prevent the model from overfitting. After 5 epochs, early stopping stopped the training since no further improvement was seen in the validation loss. This helped maintain a balance between training on the given data and ensuring that the model generalizes well to new data.
- Model Performance
    - The model resulted in a test accuracy of 75.79**%**, which is consistent with the validation accuracy during training. This suggests that the model can correctly predict sentiment with a relatively high degree of accuracy.

G: Recommendations

Based on the analysis of the model's performance, there are some recommended actions to take. The first being the trained model can now be deployed to automatically classify customer reviews or feedback in real-time. This can help businesses quickly assess customer sentiment toward their products or services and allow faster decision-making and more targeted responses to customer feedback. The next recommendation is to improve the model accuracy because while 75.79% is solid, it leaves room for improvements. This could be through hyperparameter tuning, regularization to reduce overfitting and improve validation, or experimenting with more complex architectures such as LSTM or GRU that are more specialized for sequence data to capture long-term dependencies in text. The last recommendation would be to continuously monitor the model performance. As new data becomes available, it's important to periodically re-train or fine tune the model so that the predictions are consistently accurate.

H: Reporting

The files submitted with this task will be labeled as

- 'WGU_D213_Task_2.html'
- 'WGU_D213_Task_2.ipynb'

I: Sources for Third-Party Code

1. "How to Perform Sentiment Analysis with Python?" Retrieved from https://365datascience.com/tutorials/python-tutorials/sentiment-analysis-with-python/
2. "Display Deep Learning Model Training with Keras" Retrieved from https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

3. “Understanding masking & padding” Retrieved from https://www.tensorflow.org/guide/keras/understanding_masking_and_padding

J: Sources

1. “What is tokenization?” Retrieved from https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-tokenization
2. “Sentiment Analysis Using Python” Retrieved from https://www.analyticsvidhya.com/blog/2022/07/sentiment-analysis-using-python/
3. “What is Hyperparameter Tuning?” Retrieved from https://aws.amazon.com/what-is/hyperparameter-tuning/#:~:text=computationally%20intensive%20process.-,What%20are%20hyperparameters%3F,set%20before%20training%20a%20model.