A1) Research Question

"Can we utilize neural networks and natural language processing (NLP) to predict customer sentiment data to create actionable insight for a company?"

A2) Objectives

The goal of this analysis is to utilize recurrent neural networking on a combined dataset of reviews from Amazon, IMDB, and Yelp to detect patterns and make predictions that provide actionable insight for stakeholders.

A3) Neural Network Identification

Recurrent neural network (RNN) is a type of artificial neural networking utilized for sequential data. This type of neural network is characterized by the ability to memorize context over time. This type of neural network is effective for tasks that involve sequencing, such as natural language processing. These characteristics make RNN ideal for predicting text sequences and patterns within our dataset.

B1) Exploratory Data Analysis

- Unusual characters – these are characters within the data frame that are non-standard or special characters. Below is the code I utilized to detect unusual characters.

Figure 1

Detection for Unusual Characters

```
In [15]: unusual_char = df['Review']
         list_of_char = []
         for comment in unusual_char:
             for char in comment:
                 if char not in list_of_char:
                     list_of_char.append(char)
         print(list_of_char)

         ['s', 'o', ' ', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', 'd', 'x', 'j', '4',
         '5', 'z', 'q', 'k', '7', '3', '6', '8', '0', '2', '1', '9', '\t', '\n', 'é', '\x85', 'â', 'ê']
```

- Vocabulary Size – the vocabulary size in natural language processing refers to the number of unique words within a dataset.

Figure 2

Vocabulary Size

```
In [16]: text_data = df['Review'].astype(str)

         tokens = [word_tokenize(text) for text in text_data]
         all_words = [word for sublist in tokens for word in sublist]
         vocabulary = set(all_words)
         vocabulary_size = len(vocabulary)
         print("Vocabulary Size:", vocabulary_size)

         Vocabulary Size: 5400
```

- Proposed word embedding length – the embedding length within a neural network is a parameter that determines the size of vector space in which tokens are represented as dense. This is usually determined by vocabulary size, task complexity, and size of the dataset.

Figure 3

Proposed Embedding Length

```
In [18]: sequences = tokenizer.texts_to_sequences(df['Review'])
         tokenizer = Tokenizer()
         max_sequence_length = max(len(seq) for seq in sequences)
         proposed_embedding_length = int(np.ceil(max_sequence_length * 0.1))
         print(proposed_embedding_length)

         139

In [19]: proposed_embedding_length = int(round(np.sqrt(np.sqrt(len(vocabulary))), 0))
         print("Proposed Word Embedding Length:", proposed_embedding_length)

         Proposed Word Embedding Length: 9
```

- Maximum sequence length – the maximum sequence length refers to the maximum number of words allowed in a single sequence input.

Figure 4

Maximum Sequence Length

```
In [20]: sequence_lengths = [len(seq) for seq in sequences]
         max_sequence_length = np.max(sequence_lengths)
         median_sequence_length = np.median(sequence_lengths)
         min_sequence_length = np.min(sequence_lengths)
         print("Maximum Sequence Length:", max_sequence_length)
         print("Median Sequence Length:", median_sequence_length)
         print("Minimum Sequence Length:", min_sequence_length)

         Maximum Sequence Length: 1384
         Median Sequence Length: 10.0
         Minimum Sequence Length: 1
```

B2) Tokenization

Tokenization within the context of natural language processing is the process of breaking down text into smaller units, this is the goal with the tokenization of our dataset. This is a crucial step in the natural language process step as it converts raw text into a format that is more manageable for neural networks and machine learning models to process (Awan, 2023). To accomplish this with our dataset, the library natural language toolkit (NLTK) was installed. This library provides tools related to natural language processing and analysis of text. After this library was installed, the package word_tokenize and tokenizer were utilized to tokenize the data.

Figure 5

Tokenization

```
In [17]: # Applying stopwords, tokenizing, and vectorize
         stop_words= set(stopwords.words('english'))

         data_set = []
         for sentence in df.Review:
             data_set.append([word for word in word_tokenize(sentence) if word not in stop_words])

         print(data_set)
         vocab_size=5400
         x=df['Review']
         print('''\n''',x)
         tokenizer = Tokenizer(num_words = vocab_size)
         tokenizer.fit_on_texts(x)
         x = tokenizer.texts_to_sequences(x)
         print('''\n''',x)

         [['way', 'plug', 'us', 'unless', 'go', 'converter'], ['good', 'case', 'excellent', 'value'], ['great', 'jawbone'], ['tied',
         'charger', 'conversations', 'lasting', '45', 'minutesmajor', 'problems'], ['mic', 'great'], ['jiggle', 'plug', 'get', 'line',
         'right', 'get', 'decent', 'volume'], ['several', 'dozen', 'several', 'hundred', 'contacts', 'imagine', 'fun', 'sending', 'on
         e', 'one'], ['razr', 'owneryou', 'must'], ['needless', 'say', 'wasted', 'money'], ['waste', 'money', 'time'], ['sound', 'qual
         ity', 'great'], ['impressed', 'going', 'original', 'battery', 'extended', 'battery'], ['two', 'seperated', 'mere', '5', 'ft',
         'started', 'notice', 'excessive', 'static', 'garbled', 'sound', 'headset'], ['good', 'quality', 'though'], ['design', 'odd',
```

B3) Padding Process

Padding within the context of natural language processing is the process of adding fillers of various lengths to sequences to make them equal in length. This process improves the efficiency and computation within neural network models. This was accomplished by utilizing the function texts_to_sequences() and pad_sequences(). The filler, in this case zeroes were utilized, after the text sequences.

Figure 6

Single Padded Sequence

```
In [22]: from sklearn.datasets import make_classification
         import random
         n = random.randint(0, len(rev_list))
         print('"', df.Review[n], '"')
         print(padded[n])

" so there is no way for me to plug it in here in the us unless i go by a converter "
[27 49  5 ...  0  0  0]
```

B4) Categories of Sentiment

Due to our categories of sentiment being either 1 (positive) or 0 (negative), the number of categories of sentiment for our analysis is 2. Considering our categories of sentiment, the most optimal dense layer for our neural network is sigmoid. The sigmoid activation function outputs probability values between 0 and 1, making this ideal for our analysis.

Figure 7

Categories of Sentiment

```
In [23]: num_classes = 2
         model = tf.keras.Sequential([
             tf.keras.layers.Dense(num_classes, activation='sigmoid')
         ])

         print("Number of sentiment categories:", num_classes)

Number of sentiment categories: 2
```

B5) Data Preparation Steps

- Read and import the datasets into the Jupyter notebook and combine the datasets into one data frame
- Conduct data cleaning and exploratory analysis by checking for shape, missing values, vocabulary size, unusual characters, embedding length
- Clean text by lowercasing, removing punctuation, and removing unwanted characters
- Implement stopwords to remove conjunctions and word particles
- Lemmatize the data
- Tokenize the data
- Vectorize the data with integer encoding
- Add padding to either before or after the sequences
- Split the dataset into validation, training, and test sets into an 80/20 split, as this is the common partitioning utilized for machine learning (Pramoditha, 2021).

B6) Copy of Prepared Data Set

A CSV file of the prepared dataset is attached to this project

C1) Model Summary

To create our neural network model, I imported the library TensorFlow and utilized the package keras. Keras is a package in Python that is utilized to build, configure, and deploy neural networks. This package was utilized to develop our neural network model and provided us the following output from model.summary().

Figure 8

Model Summary (Shaikh, 2023)

```
In [27]: from tensorflow.keras import Sequential
         from tensorflow.keras.layers import LSTM, Dropout, Dense, Embedding
         from tensorflow.keras.callbacks import EarlyStopping
```

```
In [28]: embed_size=int(round(vocabulary_size ** (1/4))) #, 0))

         model = Sequential()
         model.add(Embedding(vocab_size, embed_size, input_shape = (X_train.shape[1],)))
         model.add(LSTM(units=60, activation='sigmoid'))
         model.add(Dense(units=1, activation='sigmoid'))

         model.compile(optimizer='adam', loss='binary_crossentropy', metrics='accuracy')
         model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 2748, 9)           48600

 lstm (LSTM)                 (None, 60)                16800

 dense_1 (Dense)             (None, 1)                 61

=================================================================
Total params: 65461 (255.71 KB)
Trainable params: 65461 (255.71 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

C2) Model Parameters

Our neural network model consists of 3 layers, which are the embedding, LSTM, and dense layers. The embedding layer is used within natural language processing to create embeddings, or numerical vector representations, for categorical variables, this layer consists of 48,600 trainable parameters. The Long Short-Term Memory (LSTM) layer assists our neural network model to process long-term dependencies in sequential data, and this layer consists of 16,800 trainable parameters. The dense layer allows our neural network model to process complex patterns and embeddings through linear and non-linear transformations, and this layer consists of 61 trainable parameters.

C3) Hyperparameters

- Activation Functions – these input non-linear transformations to the nodes within the layer of the network. The LSTM and Dense layer both utilize the sigmoid activation function. This type of activation function was utilized due to our outputs being binary, between 0 and 1.

- Nodes – these can also be referred to as neurons, are computational units that perform operations on input data. Our LSTM layer contains 60 nodes, and our dense layer contains 1 node. The number of nodes were determined by trial and error as there is no set formula or calculation for the number of nodes to utilize.
- Loss Functions – these quantify differences between predicted and true values and guide/adjust the network during the learning process detecting for incorrect labeling. The loss function for our model is binary cross-entropy due to our outputs being binary.
- Optimizer – these adjust the attribute of our neural network model based on computed gradients of our loss function. The optimizer utilized for our model is adam. This optimizer is ideal for recurrent networks as it is able to handle and adjust to the complex training dynamics of these types of networks.
- Stopping Criteria – this provides predefined conditions for our model to stop the training process. Our model utilized early stopping, which stops the training when the performance of our validation stops improving.
- Evaluation Metric – considering our neural network model is classification, I utilized accuracy, the ratio of correctly predicted values to the total number, as our evaluation metric.

D1) Stopping Criteria

As stated prior, the stopping criteria within a neural network model provides predefined conditions for our model to stop when the performance of our validation stops improving. Utilizing a stopping criterion helps to prevent overfitting within our model. An epoch is a complete cycle of our model through the training set, it is a hyperparameter that determines the number of times a model will work through the training set. Utilizing our stopping criteria, our model was run with a various number of epochs until our stopping criteria halts. This resulted

when we ran 20 epochs, the stopping criteria halted our model at 12 epochs when our validation loss did not improve.

Figure 9

Training Epoch

```
In [29]: from tensorflow.keras.callbacks import EarlyStopping

         early_stopping_monitor = EarlyStopping(patience=8)
         history = model.fit(X_train, y_train, epochs = 20, batch_size = 32, validation_split=0.3, verbose=True, callbacks=[early_stopping
```

```
Epoch 1/20
69/69 [==============================] - 91s 1s/step - loss: 0.7011 - accuracy: 0.5227 - val_loss: 0.7074 - val_accuracy: 0.470
9
Epoch 2/20
69/69 [==============================] - 96s 1s/step - loss: 0.6953 - accuracy: 0.4927 - val_loss: 0.7112 - val_accuracy: 0.470
9
Epoch 3/20
69/69 [==============================] - 101s 1s/step - loss: 0.6957 - accuracy: 0.5064 - val_loss: 0.6969 - val_accuracy: 0.47
09
Epoch 4/20
69/69 [==============================] - 101s 1s/step - loss: 0.6948 - accuracy: 0.4982 - val_loss: 0.6915 - val_accuracy: 0.52
91
Epoch 5/20
69/69 [==============================] - 103s 1s/step - loss: 0.6941 - accuracy: 0.5000 - val_loss: 0.6916 - val_accuracy: 0.52
91
Epoch 6/20
69/69 [==============================] - 109s 2s/step - loss: 0.6950 - accuracy: 0.4927 - val_loss: 0.6923 - val_accuracy: 0.52
91
Epoch 7/20
69/69 [==============================] - 104s 2s/step - loss: 0.6950 - accuracy: 0.4927 - val_loss: 0.6919 - val_accuracy: 0.52
91
Epoch 8/20
69/69 [==============================] - 106s 2s/step - loss: 0.6938 - accuracy: 0.4909 - val_loss: 0.6987 - val_accuracy: 0.47
09

Epoch 9/20
69/69 [==============================] - 110s 2s/step - loss: 0.6951 - accuracy: 0.4818 - val_loss: 0.6984 - val_accuracy: 0.47
09
Epoch 10/20
69/69 [==============================] - 107s 2s/step - loss: 0.6939 - accuracy: 0.5073 - val_loss: 0.6979 - val_accuracy: 0.47
09
Epoch 11/20
69/69 [==============================] - 109s 2s/step - loss: 0.6936 - accuracy: 0.5127 - val_loss: 0.6941 - val_accuracy: 0.47
09
Epoch 12/20
69/69 [==============================] - 113s 2s/step - loss: 0.6935 - accuracy: 0.5000 - val_loss: 0.7007 - val_accuracy: 0.47
09
```

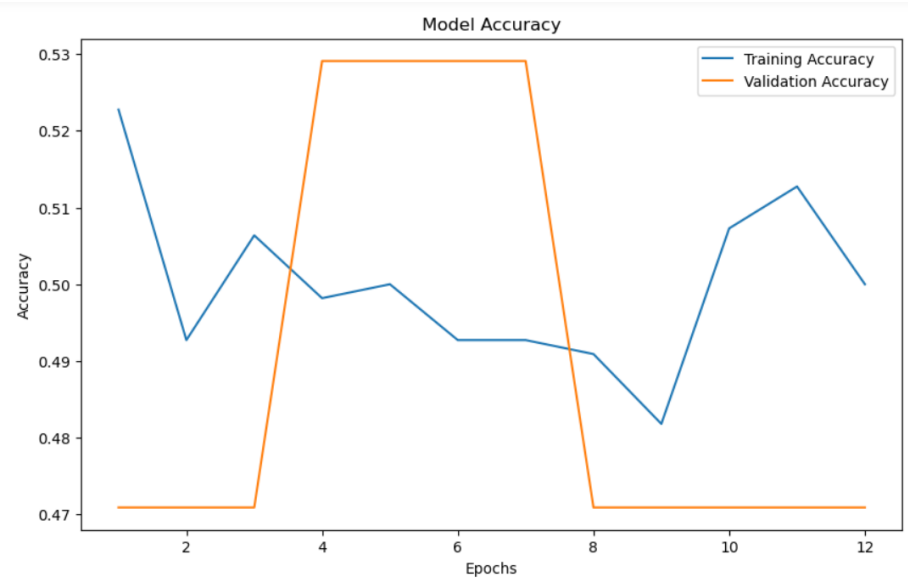
D2) Overfitting

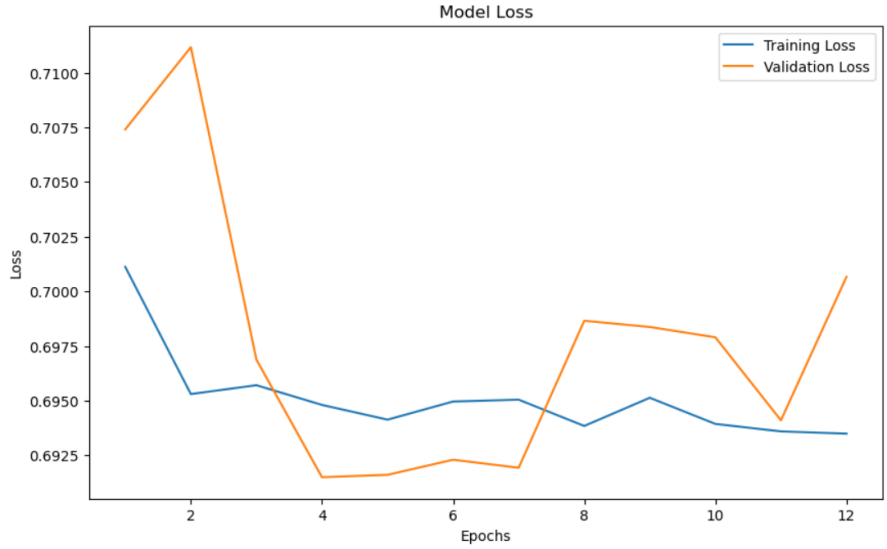
Overfitting within a neural network occurs when our model learns the training data too well and captures noise that does not generate new data. This can be observed by comparing the training set with the testing set. Thus, separating our dataset into a training and test set can help in detecting overfitting. A method to prevent overfitting is to utilize stopping criteria, which we have applied the stopping criteria of "early stopping" to our neural network model.

D3) Visualizations of Model Training Process

Figure 10

Line Graph of Loss and Evaluation






```
In [30]: model.evaluate(X_test, y_test)
         18/18 [==============================] - 7s 404ms/step - loss: 0.7007 - accuracy: 0.4709
Out[30]: [0.7006620764732361, 0.47090908885002136]
```

D4) Predictive Accuracy

Our model was processed through a various number of stopping criteria and epochs. The best predictive accuracy we achieved was 47.09 %. This indicates we were able to correctly

predict 47.09% of training outcomes. Our average loss was 0.6937. The predictive accuracy of our model is not optimal to confidently predict customer sentiment, and this could be due to many reasons. One reason could be due to our sample size, perhaps a larger sample size of 3000 would help improve our accuracy by providing a larger training set and vocabulary size for our model to train. Other reasons could be due to inefficient data preparation or neural network architecture.

E) Code for Saved Trained Network

The code for my save trained network within our neural network is attached to this project.

F) Functionality of Neural Network

As previously mentioned, our neural network was trained utilizing the customer reviews and sentiment provided by the combined data frame of our 3 datasets from Amazon, IMDB, and Yelp. The architecture of our RNN model is well suited for text classification and structured for the consideration of our binary sentiment values. Though the prediction accuracy was approximately 50%, we were still able to predict whether a user's review was positive or negative based on test classification. This will provide actionable insights for stakeholders to improve customer satisfaction.

Figure 11

Prediction

```
In [34]: sentiment_label = df.Sentiment_Value.factorize()

In [35]: def predict_sentiment(text):
             tw = tokenizer.texts_to_sequences([text])
             tw = pad_sequences(tw,maxlen=200)
             prediction = int(model.predict(tw).round().item())
             print("Predicted label: ", sentiment_label[1][prediction])

In [39]: test_sentence1 = "good case and excellent value"
         predict_sentiment(test_sentence1)

         1/1 [==============================] - 0s 21ms/step
         Predicted label:  1
```

G) Recommendations

Based on our low prediction accuracy value, I would not be confident for a company to utilize our neural network model. I would recommend providing a larger sample size to hopefully improve the accuracy of our model. I would also recommend separating the data into product-based reviews for predictions. I believe it may be difficult to accurately predict customer sentiment when combining reviews for multiple products across multiple platforms. I believe it would be more effective to analyze reviews and sentiments based on types of products, rather than across a whole platform with a numerous number of different products. I would also recommend adding variables to our dataset with specific dates for a review and products that the review is linked to.

Third-Party Code

Shaikh, J. (2023, July 31). *Build your neural network using Tensorflow*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow/

References

Awan, A. A. (2023, September 22). *What is tokenization? types, use cases, implementation*. DataCamp. https://www.datacamp.com/blog/what-is-tokenization

Pramoditha, R. (2021, November 23). *How large does the test data set need to be?*. Medium. https://medium.com/data-science-365/how-large-does-the-test-data-set-need-to-be-7181922d5dda