D213: Advanced Data Analytics - Task 2

Neural Networks and Natural Language Processing for Sentiment Analysis

Andrew Mecchi

Masters of Data Analytics, Western Governors University

The following report covers the Performance Assessment for D213 Advanced Data Analytics, Task two. This document is categorized by the questions defined in the rubric.

Part I: Research Question

A1. Summarize one research question that you will answer using neural network models and NLP techniques.

Can the sentiment of a customer review be accurately predicted as positive or negative from a trained model using Natural Language Processing (NLP) and Recurrent Neural Networks (RNN)?

A2. Define the objectives or goals of the data analysis.

The objective of the analysis is to design a reliable model to accurately predict customer sentiment from historical data. To achieve this goal, customer reviews from Amazon, Yelp, and IMDB will be used to train a neural network model whereby word frequencies will teach the model patterns of language used to help classify positive or negative sentiment. Ultimately, an accurate model will assist businesses with evaluating customer satisfaction based on the text of their reviews. Understanding the disposition of consumers is paramount for success and predicting customer satisfaction from reviews will assist businesses with the knowledge necessary to identify which products are considered favorable or unfavorable among clients.

A3. Identify a type of neural network capable of performing a text classification task that can be trained to produce useful predictions on text sequences on the selected data set.

The goal of this analysis is to design a neural network model using historical data of reviews from Amazon, IMDB, and Yelp to handle classification of future customer reviews. A neural network is a deep learning algorithm composed of multiple node layers, containing an input layer, hidden layers, and an output layer; whereby each node connects to another and has an associated weight and threshold, and depending on the output value a node, the next layer will be activated (meets threshold) or no data is passed along to the next layer if the activation doesn't meet threshold (IBM, n.d.). The structure of a neural network resembles the biological network of neurons in the human brain where network nodes are representative of neurons. For this analysis, a recurrent neural network (RNN) will be employed. Recurrent neural networks are a variant of a recursive artificial neural network in which connections between nodes make a directed cycle where outputs depend not only on current inputs, but also on the previous step's node state, thus, RNN's are often used for sentiment analysis and text classification (Data Monsters, 2017). The customer reviews from Amazon, IMDB, and Yelp will be prepared for analysis by creating uniform text strings, tokenized (convert the strings into individual words/tokens), stop words removed (words that don't add value to sentences), lemmatized (reduce word structures), and vectorized (assign numeric values for use in algorithm). A RNN will be designed to comb through the treated customer reviews and identify vector frequencies and their associated sentiments.

An effective RNN model will accurately classify customer sentiment based on the identification of frequent word patterns found in the training dataset (represented by numeric vectors). The model will also institute a bidirectional Long Short-Term Memory (BiLSTM) which is a powerful tool in modeling sequential dependencies between words and phrases by allowing input to flow in both directions, and is capable of utilizing information from both sides (Zvornicanin, 2023). The underlying relationship between customer reviews and their corresponding sentiments will be identified by the RNN model. If successful in its classification, the model can be considered for future use with businesses interested in determining product strengths and weaknesses.

Part II: Data Preparation

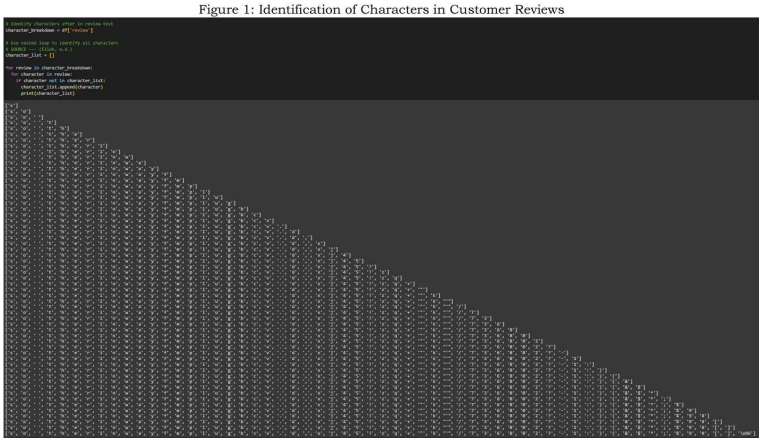
B1. Perform exploratory data analysis on the chosen data set, and include an explanation of each of the following elements:

For exploratory data analysis, three separate datasets were imported from text files; Amazon reviews, IMDB reviews, and Yelp reviews. Datasets were imported individually, each containing 500 reviews with corresponding sentiment scores and were concatenated into one dataframe totaling 1500 records. Routine exploratory data analysis was performed, assessed the data for missing values, and executed value counts before proceeding with treatment of the review text. Lastly, before treating the data for unusual characters, vocabulary size, word embedding length, and maximum sequence length, the text strings were universally converted into lowercase strings.

Presence of unusual characters (i.e. non-english characters)

Following the treatment of text to lowercase, the identification and removal of atypical characters was enacted. First, a loop was created to scan the review text for all characters used which returned a list of letters, numbers, unique characters, spaces, and non-english text (Fig. 1). After visualizing the character composition of the text, punctuation was removed using Python's regular expressions (regex) and applying a lambda function on the review column, df['review'] = df['review'].apply(lambda x : re.sub(r'[^\w\s]', '', x)).

Figure 1: Identification of Characters in Customer Reviews



The character identification loop was implemented again to view the remaining foreign character texts. Once the punctuation was removed, a function was created to treat the remaining text; whereby the reviews were tokenized, removed stop words, and lemmatized (Fig. 2). Tokenization splits the sentence structure of reviews into singular tokens, or individual words/characters. Once the function tokenizes the text, the stop words are removed, or words that are commonly used in colloquial language, but don't add any particular meaning to the information when analyzed by a computer algorithm. In addition to using the natural language toolkit's stopwords function, characters identified in the character loop were also included in stop word removal (i.e. "\x85"). Next, the text is lemmatized, a morphological analysis function which returns a word's lemma, or the base form of all its inflectional forms (Johnson, 2023). Repeating the process, the character loop was used to search for any remaining characters that would also need removal. This included the removal of numeric digits as they add no additional meaning to natural language processing as well as replacing international characters with english equivalents (Fig. 3). Following this treatment, the review text has been refined to a normalized form that will greatly reduce processing time and improve model efficiency.

Figure 2: Text Cleaning Function for Review Data

```
[19] # Create function to process text
     def clean(text):
       from nltk.corpus import stopwords
       stop_words = set(stopwords.words('english'))  # Make set of stop words
       stop_words.update('br', 'href', '\x85')  # Add unusual characters found from chacater loop to stop words
       lem = WordNetLemmatizer()  # Instantiate lemmatizer

       tokens = word_tokenize(text)  # Tokenize the text

       tok_stop = [tok for tok in tokens if tok not in stop_words]  # Remove stopwords

       tok_stop_lemma = [lem.lemmatize(tok) for tok in tok_stop]  # Lemmatize text after stopwords removed

       clean_text = ' '.join(tok_stop_lemma)

       return clean_text
```

```
[20] # Apply clean function to reviews of df

     df['clean_text'] = df['review'].apply(clean)
```

```
[21] # View clean text in df
     df
```

	review	sentiment	clean_text
0	so there is no way for me to plug it in here i...	0	way plug u unless go converter
1	good case excellent value	1	good case excellent value
2	great for the jawbone	1	great jawbone
3	tied to charger for conversations lasting more...	0	tied charger conversation lasting 45 minutesma...
4	the mic is great	1	mic great
...	...	...	...
2995	i think food should have flavor and texture an...	0	think food flavor texture lacking
2996	appetite instantly gone	0	appetite instantly gone
2997	overall i was not impressed and would not go back	0	overall impressed would go back
2998	the whole experience was underwhelming and i t...	0	whole experience underwhelming think well go n...
2999	then as if i hadnt wasted enough of my life th...	0	hadnt wasted enough life poured salt wound dra...

3000 rows × 3 columns

Figure 3: Removal of Numeric Values and Replacing Unusual Characters

```
[23] # Still numeric values, remove previous to running model
     df['clean'] = df['clean_text'].str.replace('\d+', '')

     # Still numeric values, remove previous to running model
     df['clean'] = df['clean'].str.replace('é', 'e')
     df['clean'] = df['clean'].str.replace('ê', 'e')
     df['clean'] = df['clean'].str.replace('â', 'a')
```

```
# View clean column with digits removed
df
```

	review	sentiment	clean_text	clean
0	so there is no way for me to plug it in here i...	0	way plug u unless go converter	way plug u unless go converter
1	good case excellent value	1	good case excellent value	good case excellent value
2	great for the jawbone	1	great jawbone	great jawbone
3	tied to charger for conversations lasting more...	0	tied charger conversation lasting 45 minutesma...	tied charger conversation lasting minutesmajo...
4	the mic is great	1	mic great	mic great
...	...	...	...	...
2995	i think food should have flavor and texture an...	0	think food flavor texture lacking	think food flavor texture lacking
2996	appetite instantly gone	0	appetite instantly gone	appetite instantly gone
2997	overall i was not impressed and would not go back	0	overall impressed would go back	overall impressed would go back
2998	the whole experience was underwhelming and i t...	0	whole experience underwhelming think well go n...	whole experience underwhelming think well go n...
2999	then as if i hadnt wasted enough of my life th...	0	hadnt wasted enough life poured salt wound dra...	hadnt wasted enough life poured salt wound dra...

3000 rows × 4 columns

Vocabulary size

The vocabulary size of the model was determined through the use of Tensorflow's preprocessing word tokenizer. The vocabulary size is the equivalent to the neural network model's dictionary, or, the maximum set of unique words found in the reviews. The tokenizer identified a vocabulary index of 4,360, therefore, the vocabulary size for the final model was set to include one additional word, for a total of 4,361 (Fig. 4).

Figure 4: Identify Vocabulary Size with Tokenizer

```
from tensorflow.keras.preprocessing.text import Tokenizer

# Instantiate tokenizer
tokenizer = Tokenizer(num_words = vocab_size)

# Fit tokenizer on text
tokenizer.fit_on_texts(x_clean)

# #View total count of indexed vectorized words
vocab_len = len(tokenizer.word_index) + 1
print(vocab_len)
```

```
4361
```

Proposed word embedding length

The word embeddings are words represented by dense vectors where a vector represents the projection of the word into a continuous vector space where the position of the word within the vector space is learned from the text based on the surrounding words when the word is used (Brownlee, 2021). The embedding length was determined by computing the 4th root of the number of categories, or vocabulary size (Tensorflow Team, 2017). Therefore, the word embedding length was set to 8 by taking the square root of the square root of the vocabulary size (4,360) (Fig. 5)

Figure 5: Define Maximum Embedding Length

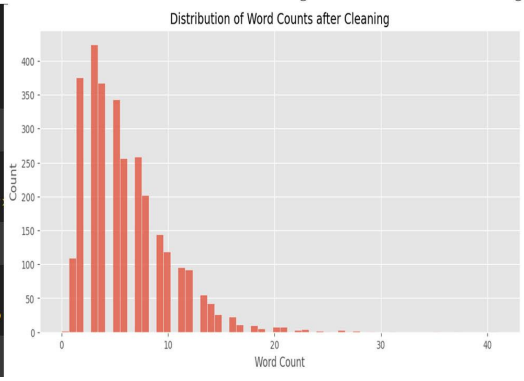
```
[54] # Define maximum embedding based on 4th root of vocab size
     max_sequence_embedding = int(round(np.sqrt(np.sqrt(vocab_len)), 0))
     max_sequence_embedding

     8
```

Statistical justification for the chosen maximum sequence length

The maximum sequence length was determined based on the range of review lengths after the text had been cleaned and normalized. Following the treatment of the review text, the word counts for the untreated and treated texts were calculated and added to the dataframe. Summary statistics were observed, maximum value identified, and a histogram plotted the spread of the word counts (Fig. 6). When observing the histogram of the word counts the distribution had a noticeable right skew, meaning, the longer length reviews were in the minority of the data.

Figure 6: Calculations of Max, Mean, and Median between Treated and Untreated Text - Histogram of Word Counts after Cleaning

```
[32] # Review word count analysis for padding sequences
     review_orig = df['word_count_review']
     review_clean = df['word_count_clean']

     # Max review length
     max_sent_orig = max(review_orig)
     max_sent_clean = max(review_clean)

     print("Max Length Original: ", max_sent_orig, '\n')
     print("Max Length Cleaned: ", max_sent_clean)

     Max Length Original:  70

     Max Length Cleaned:  41

[33] # Average review length
     avg_sent_orig = np.mean(review_orig)
     avg_sent_clean = np.mean(review_clean)

     print("Average Length Original: ", round(avg_sent_orig, 0), '\n')
     print("Average Length Cleaned: ", round(avg_sent_clean, 0))

     Average Length Original:  12.0

     Average Length Cleaned:  6.0

[34] # Average review length
     med_sent_orig = np.median(review_orig)
     med_sent_clean = np.median(review_clean)

     print("Median Length Original: ", round(med_sent_orig, 2), '\n')
     print("Median Length Cleaned: ", round(med_sent_clean, 2))

     Median Length Original:  10.0

     Median Length Cleaned:  5.0
```



Therefore, given the skewed representation, the interquartile range method was applied to identify the upper fence, meaning, the reviews above the upper fence were considered statistical outliers and would be removed from the dataframe (Fig. 7). A total of 64 reviews (~3%) were found to be outliers and removed from the sentiment analysis. The new maximum review length was set to 16, while the average review length was approximately six words (Fig. 8). Therefore, the amount of padding needed for the model is greatly reduced after the removal of the outliers.

Figure 7: Removal of Lengthy Reviews (> 16 words) and Histogram of Refined Word Counts

```
df['word_count_clean'].describe()

count    3000.000000
mean        6.121000
std         4.126423
min         0.000000
25%         3.000000
50%         5.000000
75%         8.000000
max        41.000000
Name: word_count_clean, dtype: float64

[37] # Find value of fences - mainly upper value
     q1 = 3
     q3 = 8
     iqr = q3-q1
     upper_fence = q3 + 1.5*iqr
     lower_fence = q1 - 1.5*iqr

     print("Upper Fence Value: ", round(upper_fence, 0), '\n')
     print("Lower Fence Value: ", lower_fence)

     Upper Fence Value:  16.0

     Lower Fence Value:  -4.5
```

```
[39] # Filter df for reviews with reviews with lengths between 1 and 16
     df_model = df[(df['word_count_clean'] >= 1) & (df['word_count_clean'] <= 16)]

[40] # View distribution of filtered df
     plt.figure(figsize = (12,5))
     sns.histplot(x = 'word_count_clean', data = df_model)
     plt.title("Distribution of Word Counts after Filtering <=16 Words")
     plt.xlabel("Word Count")
     plt.ylabel("Count");
```

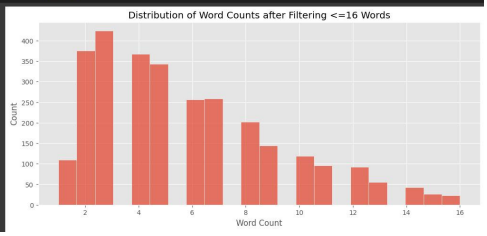
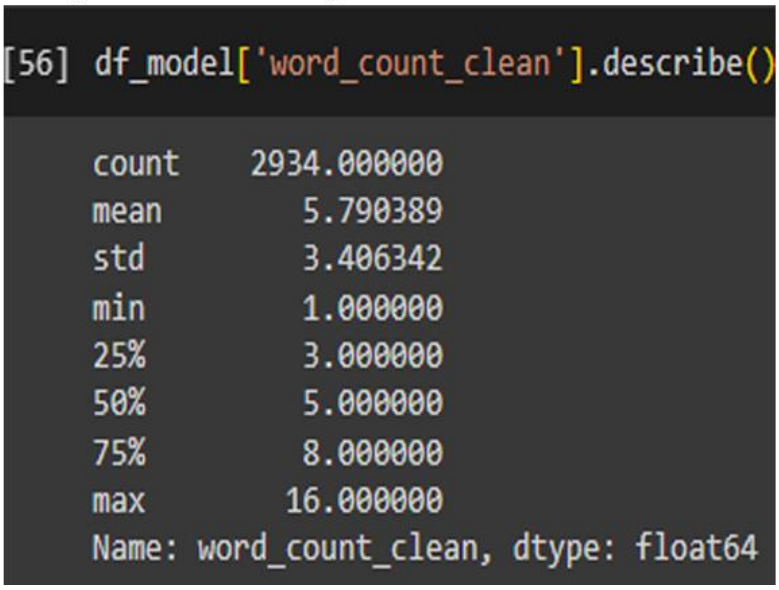


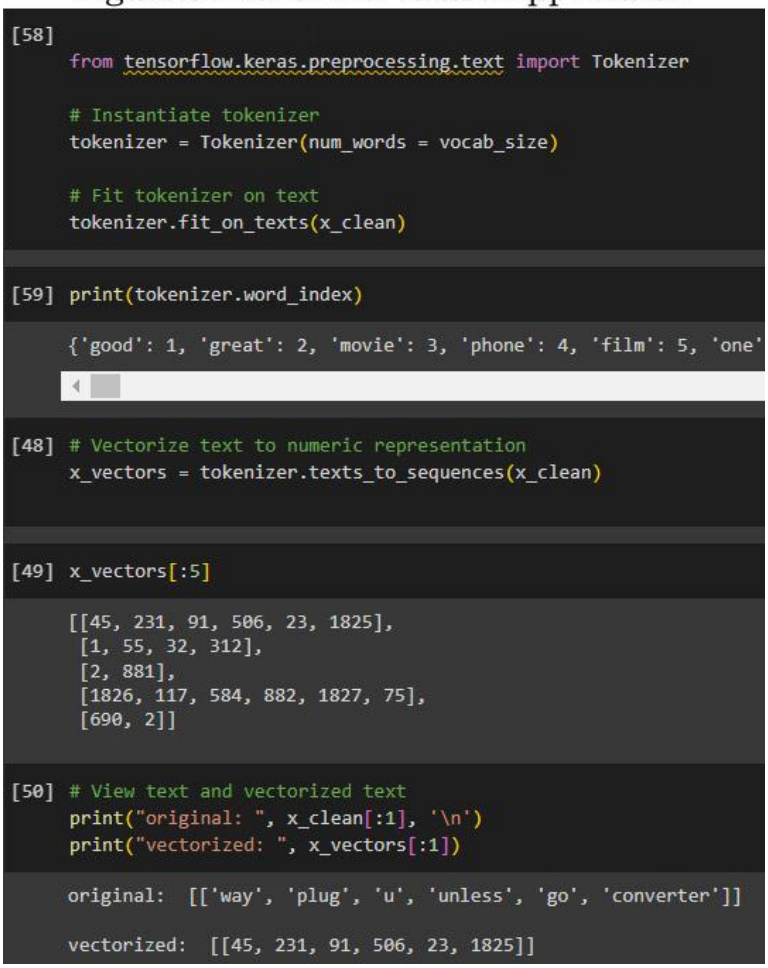
Figure 8: Summary Statistics - Clean Words



B2. Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process.

The tokenization process takes the sentences from reviews and separates each word into an individual token, for example, “the food was good” would be tokenized as “the” “food” “was” “good”, from one sentence to four tokens. Then the Tensorflow Tokenizer class vectorizes the text by turning each word into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count (Tensorflow, n.d.). The Tokenizer was instantiated, applied the fit_on_texts() to the review text, and then vectorized with text_to_sequence() resulting with the tokens represented as integers (Fig. 9). In order to run a successful RNN sentiment analysis model, all text must be converted into vectorized integers as this model cannot operate with textual values (note: there are models that can process text, but will not be used in this assessment).

Figure 9: Tokenizer Class Application



B3. Explain the padding process used to standardize the length of sequences. Include the following in your explanation:

To standardize the length of the sequences input into the model, the varying length of reviews must be padded. Previous to running the RNN model, all imputed values must be padded to have identical lengths, therefore, the pad_sequences() method is used to pad the length of all reviews to the maximum length of 16 (as determined in section B1). The method of padding was applied pre-sequence as it is better to pre-pad sequences as they are more efficient in LSTM models (Dwarampudi & Reddy, 2019). Dwarampudi and Reddy compare pre-padding to the way people talk, whereby the padded zeros are akin to silence, suggesting when person X is talking to person Y, person X waits to talk (pre-padded zeros), there's a higher chance that Y remembers most of what X said and Y's response will be more relevant to what X talked about. However, if person X said something to person Y and Y didn't reply immediately, but took some amount of time (post-padded zeros); the longer person Y waits to reply, the more context person Y loses from the conversation and more irrelevant the response will be to person X. Provided this RNN model institutes a layer of LSTM, the findings from Dwarampudi and Reddy study on

the effects of padding on LSTM's and CNN's determined the pre-padded sequences with a maximum length of 17 (maximum sentence length + 1), which can be seen in the code used in Figure 10 along with an example of a padded sequence review.

Figure 10: Padding Sequence Lengths and Padded Example

```
[60] # Define max of clean reviews
     max_sent_clean = max(df_model['word_count_clean'])

     max_sent_clean

     16

     # Import pad sequences
     from tensorflow.keras.preprocessing.sequence import pad_sequences

     # Define max length for padding
     max_length = max_sent_clean + 1

     x_pad = pad_sequences(x_vectors, maxlen = max_length, padding = 'pre', truncating = 'post')

     x_pad.shape

     (2934, 17)

[64] # View padded and vectorized review example compared to clean text
     print(x_clean[:1])
     print(x_pad[:1])

     [['way', 'plug', 'u', 'unless', 'go', 'converter']]
     [[   0    0    0    0    0    0    0    0    0    0    0   45  231   91
        506   23 1825]]
```

B4. Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network.

Based on the provided dataset, two categories define the measure of sentiment; 0 represents a negative review, whereas, 1 indicates a positive review. These same values will be the output of the final dense layer of the model defined with an activation function as sigmoid, where the output will always return a value of 0 or 1, thus being ideal for the classification determination. If the sigmoid neuron's output is larger than or equal to 0.5 the output will be 1, or a positive sentiment, while if the output is less than 0.5, the output will be 0, or negative sentiment (Kang, 2017),

B5. Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split.

1) Import review files, there are three files total; Amazon, IMDB, and Yelp. Files are either delimited with a tab (Amazon and Yelp) or a tab and a space (IMDB).

```
# Import review files for amazon, imdb, and yelp
amazon =
pd.read_csv(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytic
s\PA\Task_2\Data Sets\sentiment labeled
sentences\amazon_cells_labeled.txt', sep = "\t", header = None, index_col
= False, names = ['review', 'sentiment'], engine = 'python')
```

```
imdb =
pd.read_csv(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytic
s\PA\Task_2\Data Sets\sentiment labeled sentences\imdb_labeled.txt', sep =
"\t | \s", header = None, index_col = False, names = ['review',
'sentiment'], engine = 'python')
yelp =
pd.read_csv(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytic
s\PA\Task_2\Data Sets\sentiment labeled sentences\yelp_labeled.txt', sep =
'\t', header = None, index_col = False, names = ['review', 'sentiment'],
engine = 'python')
```

2) Explore file info for Amazon, IMDB, and Yelp

```
# View amazon df info and first three rows
amazon.info()
print('\n')
amazon.head(3)
# View imdb df info and first three rows
imdb.info()
print('\n')
imdb.head(3)
# View yelp df info and first three rows
yelp.info()
print('\n')
yelp.head(3)
```

3) Concatenate review files into one dataframe. Confirm aggregation of datasets by printing the shape and viewing the new dataframe.

```
# Concatenate three review df into one df
df = pd.concat((amazon, imdb, yelp), ignore_index = True)

# View shape
print("Shape of data: ", df.shape)

# View df
df
```

4) Search the dataframe for null or missing values.

```
# Search for missing values
df.isnull().sum()
```

5) View value counts of sentiments, based on the data dictionary there should be an equal split of sentiment for each review set, Amazon, IMDB, Yelp (all 500/500 positive/negative split). Confirm the concatenated data frame contains an aggregate value count of 1500/1500.

```
# Get an idea of pos/neg review sentiments
df['sentiment'].value_counts()
```

6) Start treatment of text and start preparations for sentiment analysis model. Starting with converting all text from the review column to lowercase text strings.

```
# Convert text of reviews into all lowercase strings
df['review'] = df['review'].str.lower()

# View conversion of reviews
df.head(3)
```

7) Remove punctuation from the review column as punctuation does not provide any additional information relative to the RNN algorithm learning model. Using Python's regular expression (regex) for removal of punctuation. Confirm removal of punctuation by viewing the first three rows.

```
# Remove punctuation using regex
# Source --- (Geeks for Geeks, 2023)

# Remove punctuation from reviews
df['review'] = df['review'].apply(lambda x : re.sub(r'[^\w\s]', '', x))

# View removal of punctuation in reviews
df.head(3)
```

8) Comb through the review column to search for and identify all characters. Looking for unusual characters that will need to be removed or amended.

```
# Identify characters after in review text
character_breakdown = df['review']

# Use nested loop to identify all characters
# SOURCE --- (Elleh, n.d.)
character_list = []

for review in character_breakdown:
  for character in review:
    if character not in character_list:
      character_list.append(character)
      print(character_list)
```

9) Create a function for treatment of the review column after text has been formatted to lowercase and stripped of punctuation. Function starts with tokenizing the text strings using NLTK's word_tokenize, then removes stop words found in NLTK's stopwords adding breaks, URL, and \x85 (found in loop from step 8) to list of stop words for removal, and lastly lemmatize the text using the WordNetLemmatizer.

```
# Create function to process text
def clean(text):
  from nltk.corpus import stopwords
  stop_words = set(stopwords.words('english'))    # Make set of stop words
  stop_words.update('br', 'href', '\x85')  # Add unusual characters found
from character loop to stop words
  lem = WordNetLemmatizer()    # Instantiate lemmatizer

  tokens = word_tokenize(text)  # Tokenize the text

  tok_stop = [tok for tok in tokens if tok not in stop_words]    # Remove
stopwords

  tok_stop_lemma = [lem.lemmatize(tok) for tok in tok_stop]    # Lemmatize
text after stopwords removed

  clean_text = ' '.join(tok_stop_lemma)

  return clean_text
```

10) Apply cleaning function to review text and add results to a new column (clean_text) for comparison/confirmation of amended text by viewing dataframe.

```
# Apply clean function to reviews of df

df['clean_text'] = df['review'].apply(clean)

# View clean text in df
df
```

11) Run another sweep for any unusual characters that were not removed after using the clean text function in step 10.

```
# Identify characters after in clean text
cleaned_breakdown = df['clean_text']

# Use nested loop to identify all characters
```

```
character_list = []

for review in cleaned_breakdown:
  for character in review:
    if character not in character_list:
      character_list.append(character)
      print(character_list)
```

12) Character list found numeric digits that need to be removed as numeric values in the text do not translate meaningful information to the RNN algorithm. Additionally, non-english characters were found and replaced with their english equivalents. Once this step is executed, re-run the character loop from step 11 to confirm removal of numeric values and non-english characters.

```
# Still numeric values, remove previous to running model
df['clean'] = df['clean_text'].str.replace('\d+', '')

# Still numeric values, remove previous to running model
df['clean'] = df['clean'].str.replace('é', 'e')
df['clean'] = df['clean'].str.replace('ê', 'e')
df['clean'] = df['clean'].str.replace('å', 'a')
```

13) Create two new columns to represent a list of words from the original review column and the treated text column. These values reflect the word count for reviews (original) and the cleaned data set (clean text).

```
# Create column to count number of words in clean text for summary
statistics
df['word_count_review'] = df['review'].apply(lambda x: len(x.split()))
df['word_count_clean'] = df['clean'].apply(lambda x: len(x.split()))
```

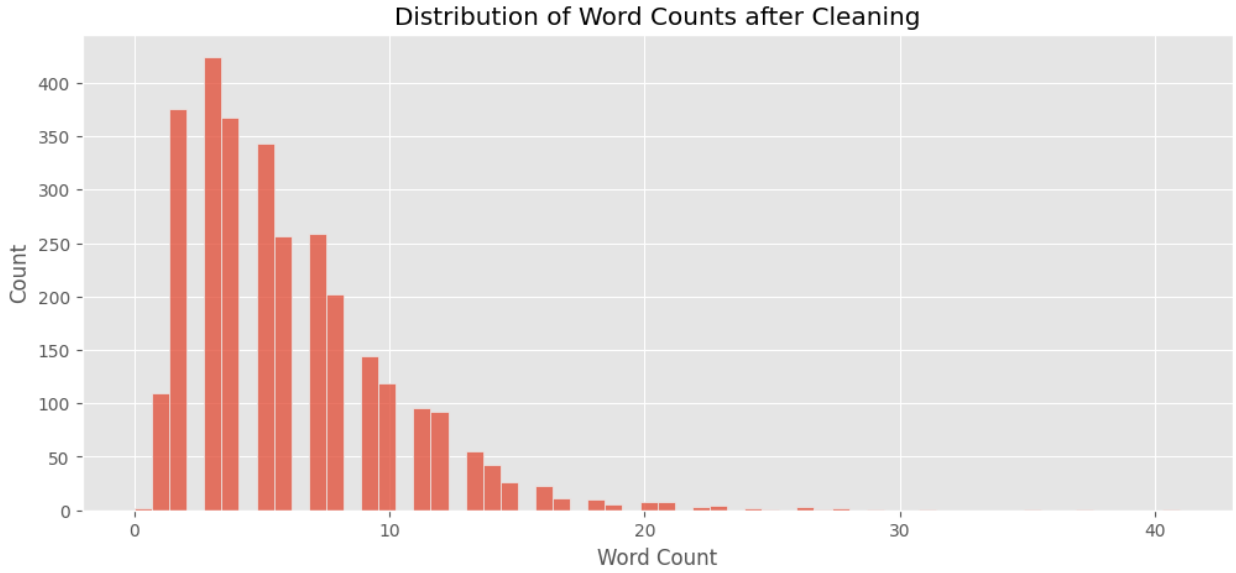
14) Calculate max, mean, and median values to compare data sets and keep note of values from clean data for model preparation.

```
# Review word count analysis for padding sequences
review_orig = df['word_count_review']
review_clean = df['word_count_clean']

# Max review length
max_sent_orig = max(review_orig)
max_sent_clean = max(review_clean)

print("Max Length Original: ", max_sent_orig, '\n')
print("Max Length Cleaned: ", max_sent_clean)
# Average review length
```

```
avg_sent_orig = np.mean(review_orig)
avg_sent_clean = np.mean(review_clean)

print("Average Length Original: ", round(avg_sent_orig, 0), '\n')
print("Average Length Cleaned: ", round(avg_sent_clean, 0))
# Average review length
med_sent_orig = np.median(review_orig)
med_sent_clean = np.median(review_clean)

print("Median Length Original: ", round(med_sent_orig, 2), '\n')
print("Median Length Cleaned: ", round(med_sent_clean, 2))
```

15) View word count distribution of the cleaned text and observe distribution of word counts.

```
# View distribution of cleaned review lengths
plt.figure(figsize = (12,5))
sns.histplot(x = 'word_count_clean', data = df)
plt.title("Distribution of Word Counts after Cleaning")
plt.xlabel("Word Count")
plt.ylabel("Count");
```



16) Noticeable right skew of word count distribution. Enlist interquartile range method to identify the upper fence (q3 + 1.5*IQR) as cutoff for model. This will limit the amount of padding needed previous to running the RNN model. Note, both q1 and q3 observed from summary statistics.

```
# Observe summary statistics of clean words
df['word_count_clean'].describe()

# Find value of fences - mainly upper value
```

```
q1 = 3
q3 = 8
iqr = q3-q1
upper_fence = q3 + 1.5*iqr
lower_fence = q1 - 1.5*iqr

print("Upper Fence Value: ", round(upper_fence, 0), '\n')
print("Lower Fence Value: ", lower_fence
```

17) View number of outliers based on results found in step 16. Isolate outliers in new dataframe named lengthy_reviews.

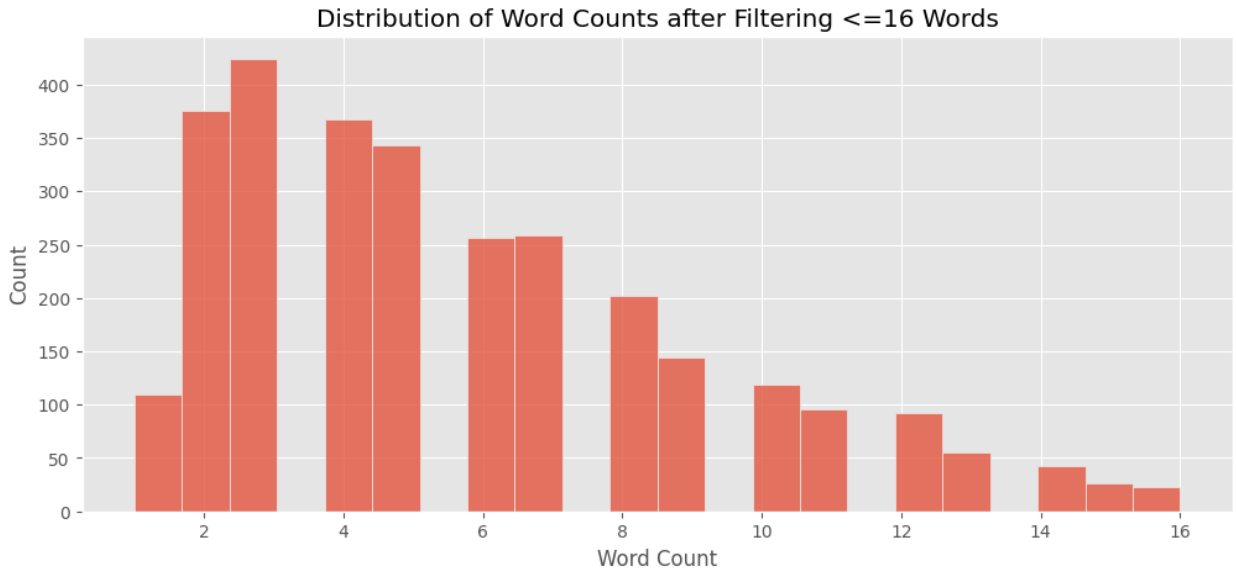
```
# summary statistics show q1 = 3 and q3 = 8 with a max value of 41
therefore outliers are present
# View counts of lengthy reviews --- reviews greater than 16 words as
determined by rounding the upper fence value for outliers
lengthy_reviews = df[df['word_count_clean'] >= 16]
lengthy_reviews['word_count_clean'].value_counts().sum()
```

18) Create a dataframe for the model by filtering reviews that have a word count between 1 and 16 and view summary statistics of the new dataframe.

```
# Filter df for reviews with reviews with lengths between 1 and 16
df_model = df[(df['word_count_clean'] >= 1) & (df['word_count_clean'] <=
16)]

# View summary statistics of new dataframe
df_model['word_count_clean'].describe()
```

19) Plot another histogram to view a more normalized distribution of word counts after removal of outliers.

```
# View distribution of filtered df
plt.figure(figsize = (12,5))
sns.histplot(x = 'word_count_clean', data = df_model)
plt.title("Distribution of Word Counts after Filtering <=16 Words")
plt.xlabel("Word Count")
plt.ylabel("Count");
```



20) Prepare cleaned data for model and train/test split. Tokenize string text found in the cleaned text columns. Will also prepare data for the appropriate format necessary for RNN model.

```
# Prepare for model, tokenize data
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))

clean_review = df_model['clean']

clean_text = []

for sentence in clean_review:
  clean_text.append([word for word in word_tokenize(sentence) if word not
in stopwords ])
```

21) Set parameters for vocabulary size, variables for cleaned and tokenized data, and create a variable for sentiment analysis.

```
# Prepare data and parameters
vocab_size = 50000
x_clean = clean_text
y = df_model['sentiment']
```

22) Employ tensorflow Tokenizer to fit_on_texts in preparation of vectorizing string text to numeric values.

```
from tensorflow.keras.preprocessing.text import Tokenizer

# Instantiate tokenizer
tokenizer = Tokenizer(num_words = vocab_size)
```

```
# Fit tokenizer on text
tokenizer.fit_on_texts(x_clean)
```

23) Convert tokenized text into numeric vectors.

```
# Vectorize text to numeric representation
x_vectors = tokenizer.texts_to_sequences(x_clean)
```

24) Define the max value of word counts and add one in preparation of padding sequences. Use Tensorflow's pad_sequences to pad all sequences with zeros thus having a uniform encoded dataset.

```
# Define max of clean reviews
max_sent_clean = max(df_model['word_count_clean'])

max_sent_clean
# Import pad sequences
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Define max length for padding
max_length = max_sent_clean + 1

x_pad = pad_sequences(x_vectors, maxlen = max_length, padding = 'pre',
truncating = 'post')

x_pad.shape
```

25) Prepare data for train/test split. The split will be 90% for training while the remaining 10% will be used for the test set. Enforce a random state for reproducibility and use stratify to equally split positive and negative sentiments among train/test sets. If not stratified, the model would learn from an imbalanced set of review sentiment which could result with a bias towards positive or negative reviews if data split unevenly.

```
# Split cleaned, vectorized, and padded data into train test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_pad, y, test_size =
0.10, random_state = 61, stratify = y)

# Confirm train test split and shape
print("Training Data Breakdown")
print(X_train.shape)
print(y_train.shape, '\n')
```

```
print("Test Data Breakdown")
print(X_test.shape)
print(y_test.shape)
```

26) Export training and test sets as csv files for performance assessment submission.

```
# Make train and test arrays into dataframes in preparation for exporting
csv files
X_train_df = pd.DataFrame(X_train)
X_test_df = pd.DataFrame(X_test)

y_train_df = pd.DataFrame(y_train)
y_test_df = pd.DataFrame(y_test)

# Export clean and prepared train and test sets
X_train_df.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Da
ta_Analytics\\PA\\Task_2\\X_train.csv', index = False)
X_test_df.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Dat
a_Analytics\\PA\\Task_2\\X_test.csv', index = False)

y_train_df.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Da
ta_Analytics\\PA\\Task_2\\y_train.csv', index = False)
y_test_df.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Dat
a_Analytics\\PA\\Task_2\\y_test.csv', index = False)
```

B6. Provide a copy of the prepared data set.

SEE ATTACHED → X_train.csv, X_test.csv, y_train.csv, y_test.csv

Part III: Network Architecture

C1. Provide the output of the model summary of the function from TensorFlow.

Figure 11: Sequential Model Code and Summary

```
# Build model using LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SpatialDropout1D, Bidirectional, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

vocab_len = len(tokenizer.word_counts) + 1
embedding_length = max_sequence_embedding
early_stop = EarlyStopping(monitor = 'val_loss', patience = 5)

# Instantiate model
model= Sequential()

model.add(Embedding(vocab_len, embedding_length, input_length = 17))
model.add(SpatialDropout1D(0.25))
model.add(Bidirectional(LSTM(64, dropout = 0.25, recurrent_dropout = 0.25)))
model.add(Dense(32, activation = 'relu', name = 'dense_layer_32'))
model.add(Dense(1, activation = 'sigmoid', name = 'dense_layer_1'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 17, 8)             34888

 spatial_dropout1d (Spatial  (None, 17, 8)             0
 Dropout1D)

 bidirectional (Bidirection  (None, 128)               37376
 al)

 dense_layer_32 (Dense)      (None, 32)                4128

 dense_layer_1 (Dense)       (None, 1)                 33

=================================================================
Total params: 76425 (298.54 KB)
Trainable params: 76425 (298.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

C2. Discuss the number of layers, the type of layers, and the total number of parameters.

The sequential model is structured with a series of five layers with 76,425 trainable parameters as seen in Figure 11 of section C1. The progression of the model starts with the embedding layer, proceeds to the spatialdropout1d layer, followed by a bidirectional long short-term memory layer, and the algorithm finishes with two dense layers, ultimately, with the final layer making the classification.

Layer 1: Embedding

The first layer of the model is the embedding layer and is an input layer responsible for turning the imputed integers into dense vectors with an input value of (None, 17, 8) with a total of 34,888 parameters. The embedding layer enables the conversion of each word into a fixed length vector of a defined size where the resultant vector is a dense one having real values instead of zeros and ones and the fixed length of the word vectors help represent words in a more efficient manner with reduced dimensions (Saxena, 2020).

Layer 2: SpatialDropout1D

The second layer of the model is a spatial dropout (one-dimensional) layer and acts as a typical dropout layer that helps prevent the model from overfitting the data. For this model, SpatialDropout1D helps promote independence between feature maps by dropping entire one-dimensional feature maps instead of individual elements; as adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers), whereas a regular dropout layer would not regularize the activations and results with a less effective learning rate, thus, spatial dropout is a more effective layer than dropout (Tensorflow, n.d.). The spatial dropout layer for this model is a dropout layer (with value of 0.25) that carries zero parameters and produces an output shape of (None, 17, 8).

Layer 3: Bidirectional(LSTM)

The third layer is a bidirectional long short-term memory (BiLSTM) layer which aims to help the model ascertain meaning from the reviews by understanding the order and sequence of words - as the order with which words are used can alter the overall meaning of the sentence. A bidirectional LSTM contains two LSTM layers, one for processing input in the forward direction and the other for processing input in the backward direction which helps natural language processing models learn the structure of preceding and following words in a sentence (Geeks for Geeks, 2023). The bidirectional LSTM layer contains a dropout and recurrent dropout argument (both with a value of 0.25) which outputs a shape of (None, 128) and contains 37,376 parameters.

Layer 4: Dense - with ‘ReLu’ activation

The fourth layer of the model is a dense layer, a layer that is deeply connected with its preceding layer, meaning, the neurons of the layer are connected to every neuron of the preceding layer and changes the dimension of the output by performing matrix vector multiplication (Verma, 2021). This is a hidden layer and outputs a shape of (None, 32) with 4,128 parameters.

Layer 5: Dense - with sigmoid activation

The fifth and final layer of the model is another dense layer, but uses a sigmoid activation for classification determination. As mentioned with layer four, this layer is deeply connected with the previous layer and performs matrix vector multiplication. This hidden layer outputs a shape of (None, 1) and contains 33 parameters.

C3. Justify the choice of hyperparameters, including the following elements:

Activation functions

Activation functions are integral parts of the RNN model where the output value(s) are transformed from the previous layer’s input. As introduced by the summary of model layers in section C2. There are activation functions in both dense layers (4 & 5), a rectified linear unit (ReLu) activation in the fourth layer and a sigmoid function in the fifth layer. Using the ReLu activation limits the number of activated nodes because negative input values result with

zeroes, meaning the neuron does not get activated, therefore, the ReLu function does not activate all the neurons at the same time which makes it more computationally efficient compared to other activations (Gupta, 2023). The sigmoid activation is used in the final layer for classification because it transforms the values to outputs of 0 or 1 and is ideal for models where the goal is to predict the probability as an output since probability of anything exists only between the range of 0 and 1 (Sharma, 2017).

Number of nodes per layer

The number of nodes in each layer vary in value, however, the first and final layer have more specific values while the remaining layers were determined through trial and error based on model accuracy. The embedding layer has very specific values that must be entered; the input dimension must be of a size one greater than the max value of the vocabulary (4,360 → 4,361), the sequence embedding size was computed as the square root of the square root of the maximum vocabulary length (8), and the input length was determined by the maximum sentence length of the treated reviews plus one. The output from the embedding layer passes through the spatial dropout layer to help prevent the model from overfitting before proceeding to the bidirectional LSTM layer. The number of nodes in the hidden layer defined by the bidirectional LSTM layer is 64 and was discovered through model trial and error. Included in the LSTM layer are a dropout and recurrent dropout parameters to further counteract potential overfitting. The nodes in the next dense layer were determined through halving the number of nodes in the previous layer, thus, 32 nodes were included in the first dense layer. At this point, the trial and error results were no longer improving model performance and/or accuracy, thus, it was determined the final dense layer be defined with a single node (best performing) and used the sigmoid activation function for classification analysis.

Loss function

The loss function helps determine the efficacy of model predictions based on the difference between the predicted values and the actual values; with a more accurate model reflecting less loss, or, minimized difference between predicted and actual values. Binary cross entropy quantifies the dissimilarity between probability distributions, aiding model training by penalizing inaccurate predictions, widely used in binary classification where the goal is to categorize data into two classes (Saxena, 2023). The loss function used for this model was binary cross entropy because the goal of the model is binary classification, predicting positive or negative sentiments from written reviews.

Optimizer

The optimizer chosen for the RNN model is the Adam optimizer, which stands for Adaptive Moment Estimation, an adaptive learning rate algorithm designed to improve training speeds in deep neural networks and reach convergence quickly by customizing each parameter's learning rate based on its gradient history allowing the neural network to learn as a whole (Agarwal, 2023). The results of the Adam optimizer are generally better than every other optimization algorithm, having faster computation time, and requires fewer parameters for tuning; therefore, Adam is recommended as the default optimizer for most of the applications and was selected to optimize this model (Gupta, 2023).

Stopping criteria

The stopping criteria for the model is enforced by Tensorflow's EarlyStopping() callback which monitors the performance of the model for every epoch on a held-out validation set during the training, and terminates the training conditional on the validation performance (Chen, 2020). The EarlyStopping() callback for this model based had a patience level set to 5, meaning, if the model showed no improvement in the validation performance evaluation metric (accuracy) after 5 epochs, the model would cease the iterative calculations. The patience level was tested at various values, but the model proved most successful with the patience level as defined.

Evaluation metric

The evaluation metric used was accuracy because the goal of the model was to successfully evaluate customer sentiments based on review text. The best way to gauge model efficacy is to calculate the accuracy of the learned model's ability to classify the unseen test set of customer sentiment.

Part IV: Model Evaluation

D1. Discuss the impact of using stopping criteria to include defining the number of epochs, including a screenshot showing the final training epoch.

The sequential model was prepared to iterate 50 epochs to provide sufficient time for the model to learn from the training dataset. However, it is possible for the validation scores to drop if the model completes the defined number of epochs. Therefore, early stopping was enforced to terminate the iterative calculations if the model showed no improvement in the accuracy of validation performance after 5 epochs (patience set to 5). The early callback resulted with the model terminating at epoch 8/50, meaning, there was no improvement in validation accuracy after epoch 4 (Fig 12).

Figure 12: Effects of EarlyStopping with a Patience Level set to 5

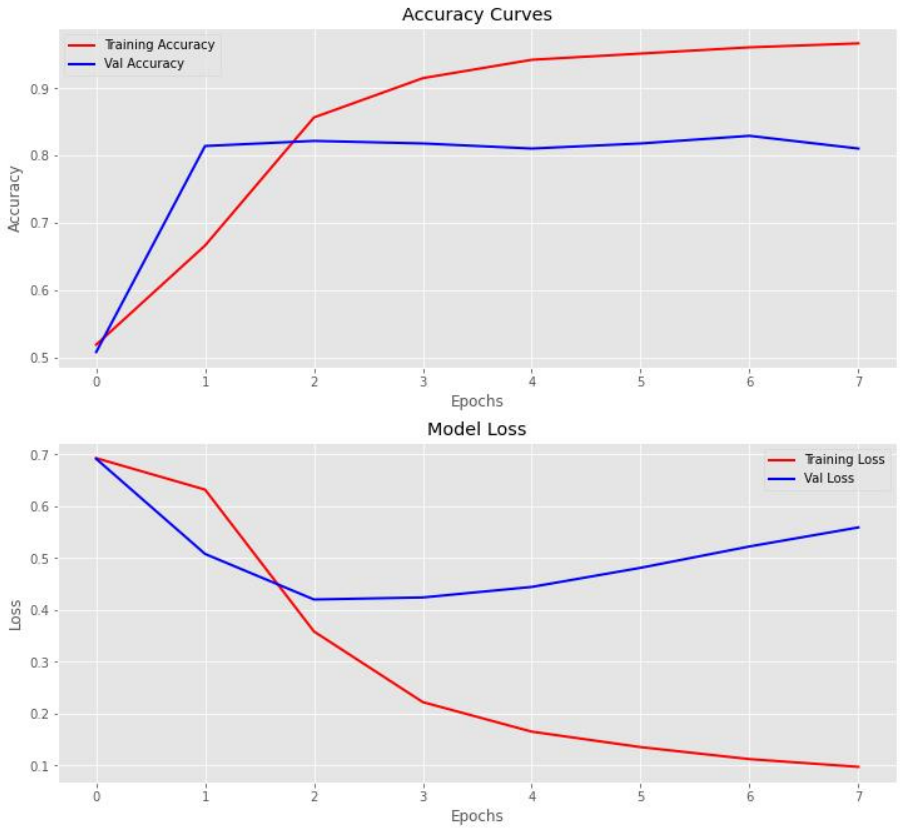
```
In [242]: history = model.fit(X_train, y_train, validation_split = 0.10, epochs = 50, batch_size = 32, callbacks = early_stop)

Epoch 1/50
75/75 [==============================] - 20s 82ms/step - loss: 0.6929 - accuracy: 0.5185 - val_loss: 0.6916 - val_accuracy: 0.5
076
Epoch 2/50
75/75 [==============================] - 5s 64ms/step - loss: 0.6322 - accuracy: 0.6662 - val_loss: 0.5081 - val_accuracy: 0.81
44
Epoch 3/50
75/75 [==============================] - 5s 71ms/step - loss: 0.3587 - accuracy: 0.8569 - val_loss: 0.4203 - val_accuracy: 0.82
20
Epoch 4/50
75/75 [==============================] - 5s 72ms/step - loss: 0.2224 - accuracy: 0.9154 - val_loss: 0.4242 - val_accuracy: 0.81
82
Epoch 5/50
75/75 [==============================] - 7s 91ms/step - loss: 0.1654 - accuracy: 0.9428 - val_loss: 0.4445 - val_accuracy: 0.81
06
Epoch 6/50
75/75 [==============================] - 6s 81ms/step - loss: 0.1356 - accuracy: 0.9520 - val_loss: 0.4814 - val_accuracy: 0.81
82
Epoch 7/50
75/75 [==============================] - 6s 74ms/step - loss: 0.1125 - accuracy: 0.9613 - val_loss: 0.5226 - val_accuracy: 0.82
95
Epoch 8/50
75/75 [==============================] - 6s 81ms/step - loss: 0.0976 - accuracy: 0.9672 - val_loss: 0.5593 - val_accuracy: 0.81
06
```

D2. Assess the fitness of the model and *any* actions taken to address overfitting.

When designing a machine learning model, overfitting is always a concern when splitting the data into training and test sets. Overfitting is a modeling error in statistics that occurs when a function is too closely aligned to a limited set of data points and results with a model that is useful in reference only to the training data and not unseen data (Twin, 2021). To address overfitting, the initial model design was limited to an embedding layer, dropout layer, LSTM layer, and a dense layer. Based on the initial results, additional model modifications were made to fine tune overfitting; the initial dropout layer was changed to a one-dimensional spatial dropout layer, the LSTM layer was made bidirectional, and an additional dense layer was added. The role of the including dropouts in the second layer and added dropout arguments in the bidirectional LSTM layer also help reduce overfitting. Given the limited number of records for model training, the train/test split was set to 90%/10% to provide the model with as much learning data as possible. To further augment model performance and optimization, access to more data for training would greatly improve model efficacy and further reduce potential overfitting.

D3. Provide visualizations of the model's training process, including a line graph of the loss and chosen evaluation metric.

Figure 13: Plots of Model Accuracy and Loss for Training and Validation



D4. Discuss the predictive accuracy of the trained network using the chosen evaluation metric from part D3.

The results of the model's ability to predict the test data sentiment returned with an accuracy of 77.55% and a loss value of 0.697 (Fig. 14). To further analyze model performance, a confusion matrix was used to assess precision, recall, and sensitivity (Fig. 15). The precision calculates the ratio of positive predictions compared to the true number of positive reviews; and returns a value of 0.7597. Recall looks at all the positive cases and informs the model's ability to successfully predict positive reviews while sensitivity evaluates the model's ability to predict negative reviews. The model returned a recall value of 0.8014 and sensitivity score of 0.7500 (Fig. 16).

Figure 14: Model Evaluation Accuracy and Loss

```
In [244]: # Evaluate model
          eval = model.evaluate(X_test, y_test, verbose = 0)

          # Isolate accuracy and loss metrics
          acc = round(eval[1]*100, 2)
          loss = round(eval[0], 3)

          # View metrics
          print("Test Accuracy: ", acc, '\n')
          print("Test Loss: ", loss)

          Test Accuracy:  77.55

          Test Loss:  0.697
```

Figure 15: Confusion Matrix of Model Performance

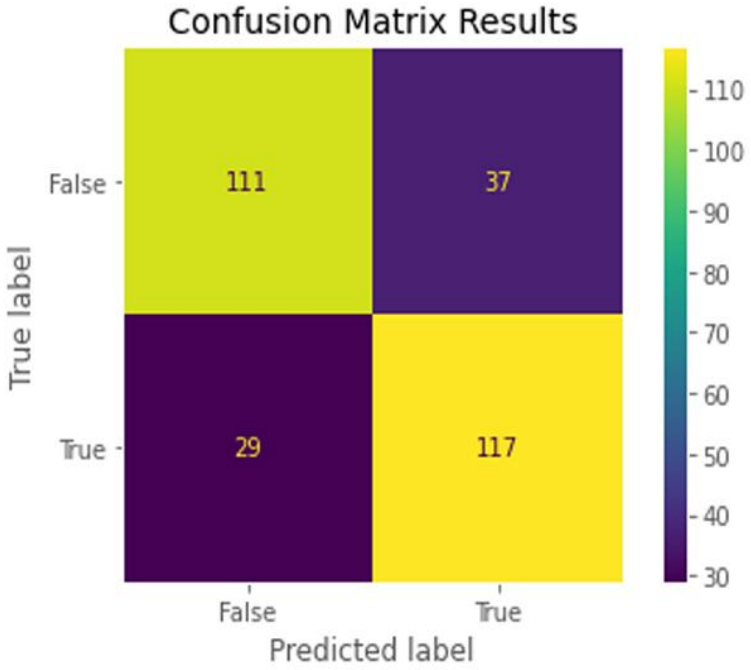


Figure 16: Performance Calculations: Accuracy, Precision, Sensitivity, Specificity

```
In [250]: # Performance Calculations - Accruacy, Precision, Sensitivity/Recall, Specificity
          # Population total
          population = tp + tn + fp + fn


          # Accuracy calculation --- sum true pos + true neg / population
          # How often is the model correct?
          accuracy = (tp + tn) / population

          # Precision calcualtion --- true pos / sum of true pos + false pos
          # Of the positives predicted, what percentage is truly positive?
          precision = tp / (tp+fp)


          # Sensitivity/Recall calculation --- true positivies / sum of true pos + false neg
          # Of all the positive cases, what percentage are predicted positive? / how good is model at predicting positive cases
          sensitivity = tp / (tp + fn)

          # Specificity calculation --- true negatives / sum of true neg + false pos
          # How well the model is at prediciting negative results?
          specificity = tn / (tn + fp)


          print("Accuracy: %0.2f" % (accuracy*100) + " %")
          print("Precision:  %0.2f" % (precision*100) + " %")
          print("Sensitivity/Recall: %0.2f" % (sensitivity*100) + " %")
          print("Specificity: %0.2f" % (specificity*100) + " %")

          Accuracy: 77.55 %
          Precision:  75.97 %
          Sensitivity/Recall: 80.14 %
          Specificity: 75.00 %
```

Part V: Summary and Recommendations

E1. Provide the code you used to save the trained network within the neural network.

SEE ATTACHED → sentiment_review_model.h5

Figure 17: Saving Model to Hierarchical Format and Importing Summary View of Saved Model

```
[ ] # Save model to hierarchal format
    model.save(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytics\PA\Task_2\sentiment_reveiw_model.h5')

[ ] # For model recreation
    reanimate_model = tf.keras.models.load_model(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytics\PA\Task_2\sentiment_reveiw_model.h5')

[181] # Show the reanimated model structure
      reanimate_model.summary()

      Model: "sequential_2"
      _________________________________________________________________
       Layer (type)                Output Shape              Param #
      =================================================================
       embedding_2 (Embedding)     (None, 17, 8)             34888

       spatial_dropout1d_2 (Spati  (None, 17, 8)             0
       alDropout1D)

       bidirectional_2 (Bidirecti  (None, 128)               37376
       onal)

       dense_layer_32 (Dense)      (None, 32)                4128

       dense_layer_1 (Dense)       (None, 1)                 33

      =================================================================
      Total params: 76425 (298.54 KB)
      Trainable params: 76425 (298.54 KB)
      Non-trainable params: 0 (0.00 Byte)
      _________________________________________________________________
```

F1. Discuss the functionality of your neural network, including the impact of the network architecture.

The natural language processing (NLP) model was based on customer reviews from three sources, Amazon, IMDB, and Yelp. There were 1,000 reviews per source for a total of 3,000 records when concatenated into one dataframe while review sentiments were equally split between positive and negative. For an NLP model to run properly, the reviews had to undergo cleaning treatments that would ultimately convert the words into numeric values for a sequential model whose algorithms could extract meaning from those values. The first of the treatments was to lowercase the reviews for uniformity and strip the text of punctuation. Then a function

was applied to tokenize the text, remove the stop words (common words that provide no additional meaning), and lemmatize potential redundancies (reduce words to their root lemma). These cleaning steps conditioned the string text of the reviews to their most meaningful attributes which were then vectorized into numeric arrays that can be imputed into the sequential model and processed for sentiment analysis.

The vectorized reviews were also padded with zeros so each record would enter the sequential model with identical review lengths. The first layer of the model embedded the numeric values into dense vectors which improve model efficiency by reducing data dimensions. The second layer contained a spatial dropout which reduces overfitting by dropping strongly correlated feature maps that would otherwise result in a less effective learning rate. The following layer utilized a bidirectional long short-term memory (LSTM) to ascertain meaning from the reviews by analyzing the order and sequence of the text values. The next layer, a dense layer, applied the ReLu activation to improve efficiency by limiting the number of active nodes by converting negative input values to zeros and telling neurons not to activate. The final layer was a dense layer with sigmoid activation, ideal for binary classification because the output is a probability analysis that returns values of 0 or 1.

For model construction, it was noted that 3,000 records is a limited quantity for training a NLP model. In addition to the relatively low sample size, 64 samples were categorized as outliers as they contained reviews with sentence lengths that were above the upper range found using the interquartile range method during exploratory data analysis. These outliers were removed to prevent the model from over-padding which could affect model performance. With the outliers removed and limited records available, the majority of the data (90%, or 2,640 reviews) was used to train the model while the remaining 10% (294 reviews) was reserved for accuracy and validation. The training data was imputed into the sequential model which analyzed the vectorized text of which the model learned what sequences and frequencies of words lead to positive or negative sentiment scores. The training model scored a 75.55% accuracy score with a loss of 0.697 when predicting the withheld testing set. The model is reasonably accurate with predicting customer sentiment based on the text of their reviews and could be effectively implemented to predict sentiment on unseen reviews. The impact of this model affirms the research question and confirms that customer sentiment can be accurately predicted positive or negative from a trained NLP model.

G1. Recommend a course of action based on your results.

Based on the model's performance with an accuracy of 77.55% and loss of 0.697, I feel confident in recommending its implementation to predict positive or negative customer sentiment based on written reviews. The model provides a solid foundation with proven efficacy, however, it should continue to be improved upon as more data becomes available. The neural network was limited to learning from 2,640 reviews, but was able to return a practical model. When more data is accessible, I would also recommend that additional data be continued to be added to train and improve model performance.

Part VI: Reporting

H1. Show your neural network in an industry-relevant interactive development environment (e.g., a Jupyter Notebook). Include a PDF or HTML document of your executed notebook presentation.

SEE ATTACHED → Panopto Video Link

Panopto copy and paste:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c6b0ae7d-d66d-49d0-9d3f-b0a10121cedb

HTML file → HTML_Python_Mecchi_D213_Task_2_Sentiment_Analysis.html

CODE REFERENCES

Elleh, F. (n.d.). D213: Advanced Data Analytics - Task 2. [PowerPoint Slides]. Faculty of Masters of Data Analytics, Western Governors University. Retrieved October 15, 2023. https://docs.google.com/presentation/d/1mFU6xLWqK8I7piJg-ECEq4xaIEFgUgre/edit#slide=id.p19.

Geeks for Geeks (2023, July 11). Python: Remove punctuation from string. Geeks for Geeks. https://www.geeksforgeeks.org/python-remove-punctuation-from-string/.

LITERARY REFERENCES

Agarwal. R. (2023, September 13). Complete Guide to the Adam Optimization Algorithm. Built In. https://builtin.com/machine-learning/adam-optimization.

Brownlee, J. (2021, February 2). How to use word embedding layers for deep learning with Keras. https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Chen, B. (2020, July 28). Early Stopping in Practice: an example with Keras and TensorFlow 2.0. Towards Data Science. https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd.

Data Monsters (2017, September 26). 7 Types of Artificial Neural Networks for Natural Language Processing. Medium. https://medium.com/@datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2.

Dwarampudi M. & Reddy N. (2019 March). Effects of padding on LSTMs and CNNs. Research Gate. https://www.researchgate.net/publication/331858065_Effects_of_padding_on_LSTMs_and_CNNs.

Geeks for Geeks. (2023, June 8). Bidirectional LSTM in NLP. Geeks for Geeks. https://www.geeksforgeeks.org/bidirectional-lstm-in-nlp/.

Gupta, D. (2023, August 4). Fundamentals of Deep Learning – Activation Functions and When to Use Them?. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/.

Gupta, A. (2023, September 13). Comprehensive Guide on Deep Learning Optimizers. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=The%20results%20of%20the%20Adam,for%20most%20of%20the%20applications.

IBM. (n.d.). What are Neural Networks?. IBM. Retrieved October 15, 2023. https://www.ibm.com/topics/neural-networks.

Johnson, D. (2023, September 30). Stemming and Lemmatization in Python NLTK with Examples. Guru99. https://www.guru99.com/stemming-lemmatization-python-nltk.html.

Kang, N. (2017, June 27). Multi-Layer Neural Networks with Sigmoid Function— Deep Learning for Rookies (2). Towards Data Science. https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

Sharma, S. (2017, September 6). Activation Functions in Neural Networks. Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Saxena, S. (2020, October 3) Understanding Embedding Layer in Keras. Analytics Vidhya. https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce.

Saxena, S. (2023, September 13). Binary Cross Entropy/Log Loss for Binary Classification. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/.

Tensorflow (n.d.). tf.keras.layers.SpatialDropout1D. TensorFlow v2.14.0 documentation. https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout1D.

Tensorflow (n.d.). tf.keras.preprocessing.text.Tokenizer. TensorFlow v2.14.0 documentation. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

Tensorflow Team (2017, November 20). Introducing TensorFlow Feature Columns. Google for Developers. https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html.

Twin, A. (2021, October 22). Understanding Overfitting and How to Prevent It. Investopedia. https://www.investopedia.com/terms/o/overfitting.asp.

Verma, Y. (2021, September 19). A Complete Understanding of Dense Layers in Neural Networks. Analytics India Magazine. https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/.

Zvornicanin, E. (2023, June 8). Differences Between Bidirectional and Unidirectional LSTM. Baeldung on Computer Science. https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm.