**Sentiment Analysis of Customer Reviews**

**Tucker Atwood, WGU MSDA**

**D213, Task 2**

**A1. Research Question**

Using the provided UCI Sentiment Labeled Sentences data set, I will answer the question, **"Can a Natural Language Processing method be used to analyze and predict customer reviews of products and services?"**

**A2. Analysis Goal**

The goal of this analysis is to **develop a Natural Language Processing (NLP) model** capable of providing insight into customer sentiment regarding products and services from Amazon, Yelp, and IMDB. The model will utilize the provided data set to examine the positive and negative sentiments of customers, so that customer behavior and opinions may be better understood. This will inform stakeholders in future decisions and strategies to address potential customer concerns and to maximize customer satisfaction.

**A3. Neural Network**

A **Deep Neural Network (DNN)** with an embedding layer and multiple dense layers will be used to develop the NLP model and conduct this sentiment analysis. A DNN is capable of classifying text in meaningful ways, such as categorizing the sentiments of customer reviews and using the results to make predictions about other customer reviews. This network type was also chosen for its ability to handle sequential data, in which information depends on previous information, such as sentences of customer reviews.

The language **Python** in the **Jupyter** environment will be used in this analysis. **Tensorflow**, a set of open-source libraries designed for machine learning, will be used with **Keras**, a user-friendly API in Python that is used on top of Tensorflow, to develop this network. These libraries will ensure the creation of a neural network capable of analyzing and predicting the sentiments of customer reviews.

**B1. Exploratory Data Analysis**

Before training the NLP model, an **exploration of the provided data set** will be performed. Before this, the data and packages must be loaded and the data must be cleaned. The following steps outline this process.

* The necessary **packages** will be loaded.

In [1]:

**import** pandas **as** pd

**import** numpy **as** np

**import** os

**import** csv

**import** re

**import** nltk

**import** random

**import** matplotlib.pyplot **as** plt

**import** tensorflow **as** tf

**from** nltk.corpus **import** stopwords

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense

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

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**from** tensorflow.keras.callbacks **import** ModelCheckpoint, EarlyStopping

**from** tensorflow.keras.layers **import** Embedding, Flatten, Dense

* The version of Tensorflow will be checked to **ensure Tensorflow has been loaded**.

In [2]:

print("Tensorflow version: ",tf**.**\_\_version\_\_)

Tensorflow version: 2.16.1

* The **directory** will be set and the **relevant data will be imported**. This includes 1000 reviews each from Amazon, Yelp, and IMDB. These reviews will be **concatenated** into one dataframe, combining the three separate sets into one, which will be used for the rest of the analysis. The resulting data frame is below.

In [50]:

os**.**chdir('C:\\Users\\atwoo\\MSDA\_Directory')

amazon **=** pd**.**read\_csv('amazon\_cells\_labelled.txt',delimiter**=**'\t',header**=None**,names**=**['review','sentiment'])

yelp **=** pd**.**read\_csv('yelp\_labelled.txt',delimiter**=**'\t',header**=None**,names**=**['review','sentiment'])

imdb **=** pd**.**read\_csv('imdb\_labelled.txt',sep**=**'\t',header**=None**, quoting **=** csv**.**QUOTE\_NONE, names**=**['review','sentiment'])

df **=** pd**.**concat([amazon,yelp,imdb], ignore\_index**=True**)

print("Concatenated dataframe containing Amazon, Yelp, and IMDB Reviews: ")

df

Concatenated dataframe containing Amazon, Yelp, and IMDB Reviews:

Out[50]:

|  | **review** | **sentiment** |
| --- | --- | --- |
| **0** | So there is no way for me to plug it in here in the US unless I go by a converter. | 0 |
| **1** | Good case, Excellent value. | 1 |
| **2** | Great for the jawbone. | 1 |
| **3** | Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!! | 0 |
| **4** | The mic is great. | 1 |
| **...** | ... | ... |
| **2995** | I just got bored watching Jessice Lange take her clothes off! | 0 |
| **2996** | Unfortunately, any virtue in this film's production work was lost on a regrettable script. | 0 |
| **2997** | In a word, it is embarrassing. | 0 |
| **2998** | Exceptionally bad! | 0 |
| **2999** | All in all its an insult to one's intelligence and a huge waste of money. | 0 |

3000 rows × 2 columns

* There are **3000 total reviews**, each **labeled** with a "0" representing a negative sentiment or a "1" representing a positive sentiment.

In [51]:

print("Dataframe shape: ")

df**.**shape

Dataframe shape:

Out[51]:

(3000, 2)

* A check for **null values** in the data will be run.

In [52]:

print("Null values: ")

df**.**isnull()**.**sum()

Null values:

Out[52]:

review 0

sentiment 0

dtype: int64

* **Zero** null values are found.
* A check for **duplicates** in the data will be run.

In [55]:

print("Duplicates: ", df**.**duplicated()**.**sum())

Duplicates: 17

* There are **17 duplicate values** in the data set. A further analysis of these duplicates will be conducted.

In [56]:

print("List of duplicates: ")

df[df**.**duplicated()]

List of duplicates:

Out[56]:

|  | **review** | **sentiment** |
| --- | --- | --- |
| **285** | Great phone!. | 1 |
| **407** | Works great. | 1 |
| **524** | Works great!. | 1 |
| **543** | Don't buy this product. | 0 |
| **744** | If you like a loud buzzing to override all your conversations, then this phone is for you! | 0 |
| **748** | Does not fit. | 0 |
| **778** | This is a great deal. | 1 |
| **792** | Great Phone. | 1 |
| **892** | Excellent product for the price. | 1 |
| **896** | Great phone. | 1 |
| **1814** | I love this place. | 1 |
| **1816** | The food was terrible. | 0 |
| **1843** | I won't be back. | 0 |
| **1846** | I would not recommend this place. | 0 |
| **2363** | Definitely worth checking out. | 1 |
| **2585** | Not recommended. | 0 |
| **2788** | 10/10 | 1 |

* Most of the duplicates are the result of common phrasing of short sentences. These have been determined to be coincidental and not true duplicates. However, one duplicate does not appear to be coincidental and is likely a **true duplicate**.

In [57]:

pd**.**set\_option('display.max\_colwidth', **None**)

print("This appears to be a true duplicate: ")

df**.**loc[744]

This appears to be a true duplicate:

Out[57]:

review If you like a loud buzzing to override all your conversations, then this phone is for you!

sentiment 0

Name: 744, dtype: object

* This duplicate will be **removed** from the data set, leaving **2999 observations** to be analyzed.

In [58]:

df **=** df**.**drop(744)

df **=** df**.**reset\_index(drop**=True**)

print("Dataframe shape after dropping the duplicate: ")

df**.**shape

Dataframe shape after dropping the duplicate:

Out[58]:

(2999, 2)

With the full data set loaded and cleaned, the **exploratory data analysis** process can now be performed. The following outlines this process.

* All **unique characters** within the data set will be identified.

In [59]:

unique **=** []

**for** review **in** df['review']:

**for** character **in** review:

**if** character **not** **in** unique:

unique**.**append(character)

print("Checking for the presence of unusual characters, here are all unique characters in the dataframe: ")

print(unique)

Checking for the presence of unusual characters, here are all unique characters in the dataframe:

['S', 'o', ' ', 't', 'h', 'e', 'r', 'i', 's', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'U', 'I', 'b', 'c', 'v', '.', 'G', 'd', ',', 'E', 'x', 'j', 'T', '4', '5', 'M', 'A', 'J', 'O', 'R', 'P', 'B', 'L', '!', 'z', 'N', 'W', 'q', 'H', '+', 'V', '"', 'Y', 'D', 'F', 'k', "'", 'K', 'C', '/', '7', '3', '6', '8', '0', '2', '?', 'Z', '-', '1', ':', ')', '(', 'Q', '&', '$', '\*', ';', 'X', '%', '9', '#', '[', ']', 'é', 'ê', '\x96', '\x85', 'å', '\x97']

* All traditional English characters exist in the data set, along with **several unusual characters** such as accented letters. These characters will be **removed**. To simplify the NLP process, all letters will also be **converted into lowercase**, and all **punctuation and numerical digits** will also be **removed**.

In [60]:

df['review'] **=** df['review']**.**str**.**lower()

df['review'] **=** df['review']**.**str**.**replace(r'[^a-zA-Z\s]', '', regex**=True**)

new\_unique **=** []

**for** review **in** df['review']:

**for** character **in** review:

**if** character **not** **in** new\_unique:

new\_unique**.**append(character)

print("Updated unique characters after converting to lowercase, removing punctuation, and unusual characters: ")

print(new\_unique)

Updated unique characters after converting to lowercase, removing punctuation, and unusual characters:

['s', 'o', ' ', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', 'd', 'x', 'j', 'z', 'q', 'k', '\x85']

* There remains one unusual character in the set.

In [61]:

print("These reviews still contain an unusual character: ")

df[df['review']**.**str**.**contains(r'\x85')]

These reviews still contain an unusual character:

Out[61]:

|  | **review** | **sentiment** |
| --- | --- | --- |
| **2177** | the script iswas there a script | 0 |
| **2966** | definitely worth seeing its the sort of thought provoking film that forces you to question your own threshold of loneliness | 1 |

* This character will be specifically removed as well. Only traditional English characters remain.

In [62]:

df['review'] **=** df['review']**.**str**.**replace('\x85', '', regex**=True**)

new\_unique\_2 **=** []

**for** review **in** df['review']:

**for** character **in** review:

**if** character **not** **in** new\_unique\_2:

new\_unique\_2**.**append(character)

print("This character has now been removed. Here are the unique characters, which are all standard English characters: ")

print(new\_unique\_2)

This character has now been removed. Here are the unique characters, which are all standard English characters:

['s', 'o', ' ', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', 'd', 'x', 'j', 'z', 'q', 'k']

* **Stopwords will be removed** as they are largely irrelevant in determining sentiment.

In [63]:

stop\_words **=** set(stopwords**.**words('english'))

**def** remove\_stopwords(text):

**return** ' '**.**join([word **for** word **in** text**.**split() **if** word **not** **in** stop\_words])

df['review'] **=** df['review']**.**apply(remove\_stopwords)

print("Here is the dataframe with stopwords removed: ")

df

Here is the dataframe with stopwords removed:

Out[63]:

|  | **review** | **sentiment** |
| --- | --- | --- |
| **0** | way plug us unless go converter | 0 |
| **1** | good case excellent value | 1 |
| **2** | great jawbone | 1 |
| **3** | tied charger conversations lasting minutesmajor problems | 0 |
| **4** | mic great | 1 |
| **...** | ... | ... |
| **2994** | got bored watching jessice lange take clothes | 0 |
| **2995** | unfortunately virtue films production work lost regrettable script | 0 |
| **2996** | word embarrassing | 0 |
| **2997** | exceptionally bad | 0 |
| **2998** | insult ones intelligence huge waste money | 0 |

2999 rows × 2 columns

* To determine vocabulary size, proposed word embedding length, and maximum sequence length, a **list of the reviews will be created and tokenized by word**. A **list of the sentiments** will also be created for future purposes.

In [64]:

texts **=** df['review']**.**tolist()

labels **=** df['sentiment']**.**tolist()

In [65]:

tokenizer **=** Tokenizer()

tokenizer**.**fit\_on\_texts(texts)

vocab\_size **=** len(tokenizer**.**word\_index)**+**1

print("Vocabulary Size: ", vocab\_size)

Vocabulary Size: 5178

* Using the tokenization process, the **vocabulary size was determined to be 5178**. This indicates the **total number of unique words** that exists in the cleaned and prepared dataframe, with unusual characters, punctuation, numerical digits, and stopwords removed.

In [66]:

embed\_length **=** int(round(np**.**sqrt(np**.**sqrt(5278)),0))

print("Proposed Word Embedding Length (fourth root of vocab size): ", embed\_length)

Proposed Word Embedding Length (fourth root of vocab size): 9

* Using the rule of thumb of the **fourth root of the vocabulary size**, the **proposed word embedding length was determined to be 9**. This indicates the **dimension of the vector** that the words will be embedded into during the creation of the model.

In [96]:

sequences **=** tokenizer**.**texts\_to\_sequences(texts)

max\_length **=** max(len(seq) **for** seq **in** sequences)

print("Maximum Sentence Length: ",max\_length)

Maximum Sentence Length: 41

* The longest sentence within the reviews was found to be **41 words long**. This information will be used to set the **maximum sequence length** for the model. This ensures that all relevant words for all reviews will be included in the model; all sequences shorter than this maximum length will be padded to create sequences of equal length for all reviews.

**B2. Tokenization**

The **tokenization process** was performed in section B1 to assist in data exploration. The **goal** of this process was to break down the customer reviews into separate words, which were then used to create a word index where each word was assigned to a number. These numbers were then converted to sequences, which is an essential part of preparing the data to be input into the eventual model.

* The **tokenization code** is included in section B1, but is replicated below:

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(texts)

sequences = tokenizer.texts\_to\_sequences(texts)

* The **package used** to normalize the text during this process was the Tokenizer package from keras.preprocessing.text. This package was imported in section B1 and also replicated below:

from tensorflow.keras.preprocessing.text import Tokenizer

* An example of the word index is included by summarizing the **first 10 words** and their associated numerical assignment:

In [68]:

print("First 10 words in the word index:")

num\_words **=** 10

**for** word, index **in** tokenizer**.**word\_index**.**items():

print(f"{word}: {index}")

num\_words **-=** 1

**if** num\_words **==** 0:

**break**

First 10 words in the word index:

good: 1

great: 2

movie: 3

phone: 4

film: 5

one: 6

food: 7

like: 8

place: 9

time: 10

* An example of the word index being used to **reconstruct a sentence** using its tokenized sequence is provided. A reminder that stopwords have been removed, so the sentence does not make complete grammatical sense, but the general sentiment is apparent. Note that "good" was shown above to be associated with "1" in the word index, and is also included in the reconstructed sentence.

In [106]:

reverse **=** {index: word **for** word, index **in** tokenizer**.**word\_index**.**items()}

**def** sequence\_to\_text(sequence):

words **=** [reverse**.**get(index, '') **for** index **in** sequence]

**return** ' '**.**join(words)

random**.**seed(1)

randseq **=** random**.**choice(sequences)

print("Random Sequence:", randseq)

sen **=** sequence\_to\_text(randseq)

print("Reconstructed Sentence:", sen)

Random Sequence: [24, 133, 12, 49, 1, 22]

Reconstructed Sentence: love camera really pretty good quality

**B3. Padding Process**

The NLP model will require all sequences to be of equal length, so the **sequences will be padded** to the maximum sequence length, determined in section B1 to be 41.

* The padding will take place **after each sequence**, which means values of 0 will be added to the end of each sequence until the sequence reaches a length of 41.

In [69]:

x\_data **=** pad\_sequences(sequences, maxlen**=**max\_length, padding **=** 'post')

y\_data **=** np**.**array(labels)

print("The padded sequences: ")

print(x\_data)

The padded sequences:

[[ 45 265 95 ... 0 0 0]

[ 1 69 35 ... 0 0 0]

[ 2 939 0 ... 0 0 0]

...

[ 491 641 0 ... 0 0 0]

[5177 13 0 ... 0 0 0]

[ 934 805 924 ... 0 0 0]]

* A **screenshot of a random padded sequence** is provided. All non-zero values represent words in the initial sentence, while all 0 values have been added to the end of the sequence to ensure a length of 41.

In [70]:

np**.**random**.**seed(2)

print("Screenshot of a single padded sequence: ")

print(x\_data[np**.**random**.**randint(0, len(x\_data))])

Screenshot of a single padded sequence:

[4450 4451 4452 1878 327 4453 4454 1233 577 311 85 1410 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]

**B4. Categories of Sentiment**

The model in this analysis will attempt to determine whether customer reviews are positive or negative; these will be the **two categories of sentiment** for reviews to be categorized within. The data set was provided with each review labeled with a "0" to represent a negative review and a "1" to represent a positive review. Below is confirmation that two unique values exist in the sentiment column.

In [71]:

print("Categories of Sentiment: ", df['sentiment']**.**nunique())

Categories of Sentiment: 2

Since the categories of sentiment are binary, a **sigmoid function** will be used as the **activation function** for the final dense layer of the network. Sigmoid is a logistic-type function with outputs expressed in a range from 0 to 1, which makes it useful in classifying binary categories. This function is discussed further in section C3.

In [72]:

activation **=** 'sigmoid'

print("Activation function for final dense layer: ", activation)

Activation function for final dense layer: sigmoid

**B5. Data Preparation**

In summary, the following steps were conducted to prepare this data for analysis:

* The data was imported and concatenated.
* Null values were checked for, and none were found.
* Duplicate values were checked for. Most were determined to be coincidental; one was determined to be a true duplicate. This observation was removed.
* All non-traditional English characters, punctuation, and numerical digits were removed, leaving only traditional English characters. These were also converted to lowercase.
* Stopwords were removed from the data.
* The data was tokenized into words. Each word was assigned to a number in a word index, and sequences were created representing each sentence.
* All sequences were padded as necessary to ensure equal length.

The following data preparation step will now be conducted:

* The padded sequences and their corresponding labeled sentiments will be **split into training, test, and validation sets**. These sets will be split using the **industry standard split size, 50-25-25**. This indicates that:
* 50% of the prepared data will be used for the training set, which will be used to create the model.
* 25% of the prepared data will be used for the test set, which will be used to evaluate the predictive accuracy of the model.
* 25% of the prepared data will be used for the validation set, which will be used during training to evaluate the performance of the model and determine hyperparameter values.
* The split is performed below, with the size of each set provided.

In [75]:

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(x\_data, y\_data, test\_size**=**0.5, random\_state**=**1)

x\_test, x\_val, y\_test, y\_val **=** train\_test\_split(x\_test, y\_test, test\_size**=**0.5, random\_state**=**1)

print("Training size: ", x\_train**.**shape)

print("Testing size: ", x\_test**.**shape)

print("Validation size: ", x\_val**.**shape)

Training size: (1499, 41)

Testing size: (750, 41)

Validation size: (750, 41)

**B6. Prepared Data Sets**

A CSV file for each of the following is attached to this submission. They are summarized as follows:

* "x\_train.csv" contains the padded sequences of the training set.
* "x\_test.csv" contains the padded sequences of the test set.
* "x\_val.csv" contains the padded sequences of the validation set.
* "y\_train.csv" contains the sentiment labels of the training set.
* "y\_test.csv" contains the sentiment labels of the test set.
* "y\_val.csv" contains the sentiment labels of the validation set.
* The CSV files are created and saved below.

In [76]:

pd**.**DataFrame(x\_train)**.**to\_csv('x\_train.csv')

pd**.**DataFrame(x\_test)**.**to\_csv('x\_test.csv')

pd**.**DataFrame(x\_val)**.**to\_csv('x\_val.csv')

pd**.**DataFrame(y\_train)**.**to\_csv('y\_train.csv')

pd**.**DataFrame(y\_test)**.**to\_csv('y\_test.csv')

pd**.**DataFrame(y\_val)**.**to\_csv('y\_val.csv')

**C1. Model Summary**

The **NLP model** is defined, compiled, and fitted below. A discussion of the model is provided in further sections.

In [111]:

np**.**random**.**seed(1)

tf**.**random**.**set\_seed(1)

random**.**seed(1)

model **=** tf**.**keras**.**Sequential([

tf**.**keras**.**layers**.**Embedding(vocab\_size, embed\_length),

tf**.**keras**.**layers**.**GlobalAveragePooling1D(),

tf**.**keras**.**layers**.**Dense(100, activation **=** 'relu'),

tf**.**keras**.**layers**.**Dense(50, activation **=** 'relu'),

tf**.**keras**.**layers**.**Dense(1, activation **=** 'sigmoid')])

loss **=** 'binary\_crossentropy'

optimizer **=** 'adam'

early\_stopping\_monitor **=** EarlyStopping(monitor**=**'val\_loss', patience**=**2)

model**.**compile(optimizer **=** optimizer, loss **=** loss, metrics **=** ['accuracy'])

print("Fitting the initial keras model: ")

history **=** model**.**fit(x\_train, y\_train,

batch\_size**=**32,

epochs**=**20,

validation\_data**=**(x\_val, y\_val),

callbacks**=**[early\_stopping\_monitor],

verbose**=True**)

Fitting the initial keras model:

Epoch 1/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **2s** 13ms/step - accuracy: 0.4722 - loss: 0.6934 - val\_accuracy: 0.5000 - val\_loss: 0.6931

Epoch 2/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.5233 - loss: 0.6928 - val\_accuracy: 0.5000 - val\_loss: 0.6929

Epoch 3/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.5279 - loss: 0.6920 - val\_accuracy: 0.5013 - val\_loss: 0.6917

Epoch 4/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.5359 - loss: 0.6884 - val\_accuracy: 0.5427 - val\_loss: 0.6848

Epoch 5/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.6290 - loss: 0.6704 - val\_accuracy: 0.6293 - val\_loss: 0.6488

Epoch 6/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.7552 - loss: 0.5889 - val\_accuracy: 0.6920 - val\_loss: 0.5630

Epoch 7/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.8301 - loss: 0.4307 - val\_accuracy: 0.7373 - val\_loss: 0.5168

Epoch 8/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.8863 - loss: 0.3102 - val\_accuracy: 0.7720 - val\_loss: 0.4897

Epoch 9/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.9038 - loss: 0.2421 - val\_accuracy: 0.8067 - val\_loss: 0.4568

Epoch 10/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.9266 - loss: 0.2020 - val\_accuracy: 0.8040 - val\_loss: 0.4612

Epoch 11/20

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.9515 - loss: 0.1705 - val\_accuracy: 0.8093 - val\_loss: 0.4718

In [112]:

random**.**seed(1)

model2 **=** tf**.**keras**.**Sequential([

tf**.**keras**.**layers**.**Embedding(vocab\_size, embed\_length),

tf**.**keras**.**layers**.**GlobalAveragePooling1D(),

tf**.**keras**.**layers**.**Dense(100, activation **=** 'relu'),

tf**.**keras**.**layers**.**Dense(50, activation **=** 'relu'),

tf**.**keras**.**layers**.**Dense(1, activation **=** 'sigmoid')])

early\_stopping\_monitor **=** EarlyStopping(monitor**=**'val\_loss', patience**=**2)

model2**.**compile(optimizer **=** optimizer, loss **=** loss, metrics **=** ['accuracy'])

print("Running the updated keras model with less epochs to reduce overfitting: ")

history2 **=** model2**.**fit(x\_train, y\_train,

batch\_size**=**32,

epochs**=**8,

validation\_data**=**(x\_val, y\_val),

callbacks**=**[early\_stopping\_monitor],

verbose**=True**)

Running the updated keras model with less epochs to reduce overfitting:

Epoch 1/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **2s** 6ms/step - accuracy: 0.4722 - loss: 0.6934 - val\_accuracy: 0.5000 - val\_loss: 0.6931

Epoch 2/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 3ms/step - accuracy: 0.5233 - loss: 0.6928 - val\_accuracy: 0.5000 - val\_loss: 0.6929

Epoch 3/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 3ms/step - accuracy: 0.5279 - loss: 0.6920 - val\_accuracy: 0.5013 - val\_loss: 0.6917

Epoch 4/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.5359 - loss: 0.6884 - val\_accuracy: 0.5427 - val\_loss: 0.6848

Epoch 5/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.6290 - loss: 0.6704 - val\_accuracy: 0.6293 - val\_loss: 0.6488

Epoch 6/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.7552 - loss: 0.5889 - val\_accuracy: 0.6920 - val\_loss: 0.5630

Epoch 7/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.8301 - loss: 0.4307 - val\_accuracy: 0.7373 - val\_loss: 0.5168

Epoch 8/8

**47/47** ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step - accuracy: 0.8863 - loss: 0.3102 - val\_accuracy: 0.7720 - val\_loss: 0.4897

The **output of the model summary** created with Tensorflow is provided below.

In [42]:

print("Summary of the final model: ")

model2**.**summary()

Summary of the final model:

**Model: "sequential\_2"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ embedding\_2 (Embedding) │ (None, 41, 9) │ 46,602 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ global\_average\_pooling1d\_2 │ (None, 9) │ 0 │

│ (GlobalAveragePooling1D) │ │ │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_6 (Dense) │ (None, 100) │ 1,000 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_7 (Dense) │ (None, 50) │ 5,050 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_8 (Dense) │ (None, 1) │ 51 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 158,111 (617.62 KB)

**Trainable params:** 52,703 (205.87 KB)

**Non-trainable params:** 0 (0.00 B)

**Optimizer params:** 105,408 (411.75 KB)

**C2. Layers**

A discussion of each layer included in the network is provided as follows:

* **Layer 1** is an **embedding** layer with **46,602 parameters**. This layer converts the tokenized sequences into vectors with a dimension equal to the embedding length, which was determined to be 9. The number of parameters is determined by multiplying the vocabulary size, 5178, by the embedding length, giving 46,602 parameters for this layer.
* **Layer 2** is a **global average pooling** layer with **0 parameters**. This layer calculates the average of each feature map, which can then be fed into further layers with more efficiency, potentially reducing overfitting and reducing dimensionality. Since this performs a fixed operation, there are no parameters in this layer.
* **Layer 3** is a **dense** layer with **1000 parameters**. In this layer, nodes (discussed in section C3) from the previous pooling layer are connected to new nodes using the relu function, which is discussed in section C3. The number of parameters is determined by multiplying the number of nodes in this layer by 1 more than the number of nodes in the previous layer. In this case, there are 100 nodes in the given layer and 9 nodes in the previous layer, so the number of parameters is 100 \* (9+1) = 1000.
* **Layer 4** is a **dense** layer with **5050 parameters**. This functions much like the previous layer, providing another pass through the relu function to help the model learn complex behavior in the data. The number of parameters is calculated the same way: 50 \* (100 + 1) = 5050.
* **Layer 5** is a **dense** layer with **51 parameters**. This provides the final prediction of customer review sentiment using the sigmoid function discussed in section B4. The number of parameters is calculated the same way: 1 \* (50 + 1) = 51.
* Altogether, the network contains **5 layers: 1 embedding, 1 global average pooling, and 3 dense**, with **52,703 trainable parameters** and **105,408 optimizer parameters**, for a total of **158,111** parameters.

The general understandings and interpretations here were assisted by usage of the following resource: <https://www.tensorflow.org/api_docs>

**C3. Hyperparameters**

The relevant hyperparameters used in the NLP model are listed and discussed as follows:

* **Activation functions:**
  + The **Rectified Linear Unit (relu)** activation function was used for Layers 3 and 4, both of which were dense layers. This function was chosen to help express nonlinear aspects of the data, which increases the model's ability to describe complex relationships. The output of the relu function is decided by a piecewise function: if positive, the output is the same as the input; if negative, the output is zero. The relu function is the most common choice for hidden layers due to its efficiency and better performance over other activation functions.
  + This interpretation was assisted by usage of the following resource: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks>
  + The **Sigmoid** activation function was used for Layer 5, the final output dense layer. Sigmoid is a logistic-type function with outputs expressed in a range from 0 to 1, which makes it useful in classifying binary categories. The output of the sigmoid function gradually increases to 0, then rapidly increases until it approaches 1. This pattern allows inputs to be "pushed" toward either 0 or 1, which is especially useful when trying to predict a binary outcome, such as the customer review sentiments in this analysis.
  + This interpretation was assisted by usage of the following resource: <https://builtin.com/machine-learning/sigmoid-activation-function>
* **Nodes per layer:**
  + **Layer 3 has 100 nodes**. This can be thought of as the number of neurons, or units, within the layer connecting it to the preceding and proceeding layers. This value was chosen as a standard starting place, and through experimentation with other values it was determined that 100 was satisfactory in achieving predictive accuracy in the final model.
  + **Layer 4 has 50 nodes**. As with the previous layer, this value was chosen as a standard starting place, and through trial-and-error was also determined to be satisfactory in achieving predictive accuracy in the final model.
  + **Layer 5 has 1 node**. There is only one node in this layer because the sigmoid activation function provides one output, which is expressed as a probability from 0 to 1. Values closer to 0 are relayed as predictions of 0, a negative sentiment, while values closer to 1 are relayed as predictions of 1, a positive sentiment.
* **Loss function:** The **binary\_crossentropy** loss function was used for this model. This was chosen because the goal of the model was to predict a binary outcome, for which the binary\_crossentropy function is specifically meant to perform.
* **Optimizer:** The **adam** optimizer was used for this model. This was chosen because it is capable of iteratively updating weights in the network based on the training data. It is straightforward, efficient, and combines many of the best aspects of other similar algorithms.
  + This interpretation was assisted by usage of the following resource: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning>
* **Stopping criteria:** An **early stopping** method was included to prevent overfitting. The method had a patience value of 2 and monitored the validation loss. This stops the training of the model once its validation loss is no longer improving. The patience value determines how many epochs the model produces without improving before it is stopped. In the initial model, 20 epochs were slated to be run; however, the patience value of 2 stopped the model after 2 successive epochs did not improve the validation loss. As can be seen in the output of the initial model, in epoch 9 the validation loss reached its lowest value, then increased in epochs 10 and 11.
* **Evaluation metric:** The metric **accuracy** was used to evaluate the model's performance. This represents the percentage of correctly predicted sentiments in the customer reviews. Accuracy was used to evaluate the training set to determine how well it was learning the patterns in the data; in the validation set to help adjust hyperparameters and make decisions during the training process; and in the test set to evaluate how well the model may predict new, unseen data.

**D1. Impact of Stopping Criteria**

As discussed in section C3, an **early stopping method with a patience value of 2 and a monitor metric of validation loss** was used in the training of the model. This affected the number of epochs the model ran by stopping the training after validation loss stopped improving. The initial number of epochs specified was 20, but the stopping criteria allowed only 11 epochs to be run on the initial model. As will be discussed in section D2, however, there was still evidence that the model was overfitting, so a second model was trained with only 8 epochs to prevent significant overfitting. A summary of the **final training epoch** of that second model is summarized below:

In [113]:

print("Final Epoch Metrics:")

print(" - Training accuracy: 0.8863")

print(" - Training loss: 0.3102")

print(" - Validation accuracy: 0.7720")

print(" - Validation loss: 0.4897")

Final Epoch Metrics:

- Training accuracy: 0.8863

- Training loss: 0.3102

- Validation accuracy: 0.7720

- Validation loss: 0.4897

**D2. Fitness of the Model and Fixing Overfitting**

In the initial training of the model, 20 epochs were specified but due to the early stopping criteria only 11 epochs were run. The following summarizes the metrics of the 11th epoch:

In [114]:

print("Initial Model, 11th Epoch:")

print(" - Training accuracy: 0.9515")

print(" - Training loss: 0.1705")

print(" - Validation accuracy: 0.8093")

print(" - Validation loss: 0.4718")

Initial Model, 11th Epoch:

- Training accuracy: 0.9515

- Training loss: 0.1705

- Validation accuracy: 0.8093

- Validation loss: 0.4718

While the accuracy values of this model were satisfactory, it was evident from these loss metrics and the preceding patterns in the training loss and validation loss that the model was overfitting, despite the stopping criteria put in place to try to prevent it. Overfitting means the model was learning the patterns in the training data too well and was not improving its ability to predict new data. This was apparent because around the 8th epoch, the validation loss began plateauing while the training loss continued to decrease.

**To address this overfitting**, a second model was trained with a lower number of specified epochs. The new number of epochs was 8, which was chosen because that was where the training loss and validation loss began deviating significantly.

The updated model was still a **satisfactory fit** for the data, with approximately 88.6% accuracy for the training set and 77.2% accuracy for the validation set, and not a significant difference in training loss and validation loss. These accuracy values suggest the model was \**sufficiently able to learn and predict customer review sentiments*.

**D3. Training Process**

The model training process can be **visualized** in the following line graphs for the loss and accuracy of both the training and validation sets. As discussed above, the model's accuracy increased and loss decreased to satisfactory values for both training and validation sets, and the separations between the metrics were stopped before becoming a larger problem.

In [116]:

plt**.**figure(figsize**=**(12, 5))

plt**.**subplot(1, 2, 1)

plt**.**plot(history2**.**history['accuracy'], label**=**'Training Accuracy')

plt**.**plot(history2**.**history['val\_accuracy'], label**=**'Validation Accuracy')

plt**.**title('Model Accuracy')

plt**.**xlabel('Epoch')

plt**.**ylabel('Accuracy')

plt**.**legend(loc**=**'lower right')

plt**.**subplot(1, 2, 2)

plt**.**plot(history2**.**history['loss'], label**=**'Training Loss')

plt**.**plot(history2**.**history['val\_loss'], label**=**'Validation Loss')

plt**.**title('Model Loss')

plt**.**xlabel('Epoch')

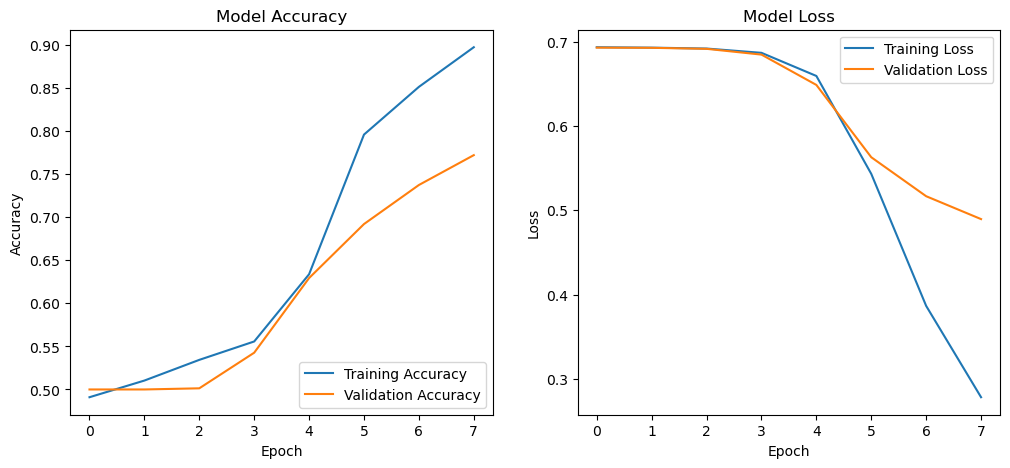
plt**.**ylabel('Loss')

plt**.**legend(loc**=**'upper right')

print("Line graphs of the model loss and accuracy, (evaluation metric):" )

plt**.**show()

Line graphs of the model loss and accuracy, (evaluation metric):



**D4. Predictive Accuracy**

To assess the **predictive accuracy** of the final model, the **accuracy and loss of the test set** are calculated below. The accuracy is approximately 73.2%, which is a satisfactory value and not significantly lower than the validation accuracy. This suggests that, given a new customer review it hasn't seen before, the model would be able to correctly predict whether it is a positive or negative review about 3/4 of the time. The loss is approximately 0.51, which a satisfactory value and is close to the validation loss in the final epoch of the model.

In [117]:

print("Using the model to evaluate the test set. Here are the metrics: ")

score **=** model2**.**evaluate(x\_test, y\_test, verbose **=** 0)

print(f'Test Accuracy: {score[1]}, Test Loss: {score[0]}')

Using the model to evaluate the test set. Here are the metrics:

Test Accuracy: 0.7319999933242798, Test Loss: 0.5099570155143738

**E. Saving the Network**

Below is the code used to save the trained model within the network.

In [45]:

model2**.**save('SentimentAnalysis.keras')

To confirm the model has been saved, it is loaded from the network.

In [46]:

print("Confirming the model has been saved: ")

loaded\_model **=** tf**.**keras**.**models**.**load\_model('SentimentAnalysis.keras')

loaded\_model**.**summary()

Confirming the model has been saved:

**Model: "sequential\_2"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩

│ embedding\_2 (Embedding) │ (None, 41, 9) │ 46,602 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ global\_average\_pooling1d\_2 │ (None, 9) │ 0 │

│ (GlobalAveragePooling1D) │ │ │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_6 (Dense) │ (None, 100) │ 1,000 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_7 (Dense) │ (None, 50) │ 5,050 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_8 (Dense) │ (None, 1) │ 51 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

**Total params:** 158,111 (617.62 KB)

**Trainable params:** 52,703 (205.87 KB)

**Non-trainable params:** 0 (0.00 B)

**Optimizer params:** 105,408 (411.75 KB)

**F. Functionality**

This analysis addressed the question, "Can a Natural Language Processing method be used to analyze and predict customer reviews of products and services?" In the process, data from Amazon, Yelp, and IMDB were imported, prepared, analyzed, and applied to the training of a neural network model, which showed satisfactory accuracy in predicting customer sentiment in new, unseen data. The **network architecture assisted in ensuring its functionality**, including: the type, structure, and functions within each layer of the model were defined such that meaningful outputs were drawn from their inputs; the stopping criteria and defined number of epochs prevented the model from overfitting; and the designated evaluation metrics reported the results in a straightforward manner. The network has been determined to be **successfully functional**.

**G. Recommended Course of Action**

Using the network developed in this analysis, **it is recommended** that stakeholders analyze and predict sentiment in customer reviews of their products and services to assist with future decisions and strategies that may address customer concerns and maximize customer satisfaction. For example, this model may identify dissatisfaction with a particular product that may not be as readily apparent, which will allow stakeholders to put forth measures to improve the product. The model may also identify particular customer interest in a product that stakeholders may use to generate adjacent interest in other products.

**H. Reporting**

This report was summarized in an Jupyter Notebook document and has been included as an .html file with the following name: “Atwood\_D213\_Task2.html.”

**I. Third-Party Code References**

WGU Courseware was used as a resource to learn the methods, concepts, and functions used to create the codes in this project, including DataCamp course tracks (datacamp.com), Dr. William Sewell’s presentations, and Dr. Elleh’s presentation materials. There are no codes that have been taken directly from any other resources.

**J. Content References**

The general understandings and interpretations of the model layers (section C2) were assisted by usage of the following resource: <https://www.tensorflow.org/api_docs>

The interpretation of the sigmoid function (section C3) was assisted by usage of the following resource: <https://builtin.com/machine-learning/sigmoid-activation-function>

The interpretation of the adam optimizer was assisted by usage of the following resource: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning>