```
Matthew Morgan
Student ID: 010471280
D213 - Advanced Data Analytics
Task 2: Sentiment Analysis
Western Governor's University
Mentor: Mandy Rasmuson
```

```
Part I: Research Question

A.  Describe the purpose of this data analysis by doing the following:

A1.  Summarize one research question that you will answer using neural network models and NLP
techniques. Be sure the research question is relevant to a real-world organizational situation
and sentiment analysis captured in your chosen data set(s).

Can we predict a user's opinion on a product or service as either positive or negative, using
previous reviews from other users?

A2.  Define the objectives or goals of the data analysis. Be sure the objectives or goals are
reasonable within the scope of the research question and are represented in the available
data.

The goal of this data analysis is to attempt to predict how a user feels about a product or
service based of the word choices that the user has chosen.

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.

Recurrent neural networks (RNNs) are a type or neural network commonly used for handling text
data and Natural Language Processing (NLP). RNN comes into play in this dataset because it is
sequential. Meaning that it is able to loop through sequential information in the input data.
This allows it to put words into context based on the words that came before it. This is
perfect for this case as we are trying to determine if reviews have positive or negative
sentiment based on the words used in previous reviews. It would make sense to use this type of
machine learning to provide accuracte predictions on user reviews by taking words into
context.
```

B1. EDA

Part I: Loading Data

In [ ]:
```
# LOADING PYTHON LIBRARIES NEEDS FOR ANALYSIS
import sys          # System Functions
import os           # View OS File Info
import pandas as pd
import numpy as np
import gzip
import re
import seaborn as sns
import nltk
import re
import matplotlib.pyplot as plt

# Scikit-Learn for predictive analysis
import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn import model_selection
from sklearn.model_selection import train_test_split
import re
import tensorflow as tf
import keras
from sklearn.metrics import confusion_matrix


import tensorflow as ft         # tensorFlow package
from tensorflow import keras    # neural network API
from keras import preprocessing

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import Dense, Embedding
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences

from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Loading text files from UCIO Sentiment Labeled Sentences Data Set
df_amz = pd.read_csv(r'C:\Users\mmorg\WGU\D213\Task 2\data\amazon_cells_labelled.txt', delimiter
df_amz.columns = ['review', 'sentiment']

df_imdb = pd.read_csv(r'C:\Users\mmorg\WGU\D213\Task 2\data\imdb_labelled.txt', delimiter='\t',h
df_imdb.columns = ['review', 'sentiment']

df_yelp = pd.read_csv(r'C:\Users\mmorg\WGU\D213\Task 2\data\yelp_labelled.txt', delimiter='\t',h
df_yelp.columns = ['review', 'sentiment']
```

In [ ]:
```
df = pd.concat([df_amz, df_imdb, df_yelp], axis=0)
```

Part II: EDA and Data Cleaning

In [ ]:
```
df.info()
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.describe()
```

In [ ]:
```
#Detect null values
print(df.isnull().sum())
```

In [ ]:
```
# Visualize distribution of sentiment data
sns.countplot(data=df, x='sentiment', palette="dark")
```

In [ ]:
```
# Find the number of positive and negative reviews
print('Number of positive and negative reviews: \n', df.sentiment.value_counts())

# Find the proportion of positive and negative reviews
print('Proportion of positive and negative reviews: \n', (df.sentiment.value_counts() / len(df)
```

In [ ]:
```
# Convert any non string values to string values
for i in range (0, len(df)-1):
    if type(df.iloc[i] ['review'])!= str:
        df.iloc[i] ['review'] = str(df.iloc[i] ['review'])
```

B1a. Presence of Unusual Characters

In [ ]:
```
# Catalog characters in review column
reviews = df['review']

list_of_characters = []
for comment in reviews:
    for character in comment:
        if character not in list_of_characters:
                list_of_characters.append(character)
print(list_of_characters)
```

In [ ]:
```
# Remove accented letters
cols = df.select_dtypes(include=[np.object]).columns
df[cols] = df[cols].apply(lambda x: x.str.normalize('NFKD').str.encode('ascii', errors='ignore'
```

In [ ]:
```
# Remove numbers
df['no_num'] = df['review'].str.replace('\d+', '')
```

In [ ]:
```
# Remove punctuation
df['no_punc'] = df['no_num'].str.replace('[^\w\s]', '')
```

In [ ]:
```
# Make all reviews lower case
df['cleaned_review'] = df['no_punc'].apply(lambda x: " ".join(word.lower() for word in x.split(
```

In [ ]:
```
# Verify absence of special characters in new column
reviews = df['cleaned_review']

list_of_characters = []
for comment in reviews:
    for character in comment:
        if character not in list_of_characters:
                list_of_characters.append(character)
print(list_of_characters)
```

B1b. Vocabulary Size

In [ ]:
```
# Create list of stop words from nltk
stop_words = stopwords.words('english')
stop_words
```

In [ ]:
```
# Remove stop words
df['stop_review'] = df['cleaned_review'].apply(lambda x: " ".join(word for word in x.split() if
df['stop_review']
```

In [ ]:
```
# Review recurring words
pd.Series(" ".join(df['stop_review']).split()).value_counts()[:40]
```

In [ ]:
```
# Create list of even more stop words relevant to this dataset
other_stop_words = ['place', 'dont', 'would', 'even', 'also', 'go', 'ive', 'im', 'get', 'could',
```

In [ ]:
```
# Remove 13 more stop words that don't have relevancy to reviews
df['stop_review'] = df['stop_review'].apply(lambda x: " ".join(word for word in x.split() if wo
```

In [ ]:
```
# Review recurring words
pd.Series(" ".join(df['stop_review']).split()).value_counts()[:40]
```

In [ ]:
```
# Create word cloud
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
stopwords = set(stopwords.words('english'))
stopwords.update(["br", "href"])
textt = " ".join(review for review in df.stop_review)
wordcloud = WordCloud(stopwords=stop_words).generate(textt)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.savefig('wordcloud1.jpg')
plt.show()
```

In [ ]:
```
# Create a list with the length of each review
length_reviews = df.stop_review.str.len()
# Look at max and min review lengths
print(max(length_reviews))
print(min(length_reviews))
```

In [ ]:
```
# Visualize shape of review lengths
plt.hist(length_reviews, range=[0, 200])
plt.title('length of review')
plt.xlabel('# of characters')
plt.ylabel('Frequency');
```

In [ ]:
```
# Print the 15 longest review lengths to see what our max review size is
line_num_words = [len(t_line) for t_line in df['stop_review']]
line_num_words.sort(reverse=False)
print(line_num_words[-15:])
```

In [ ]:
```
# Create an index so we can identify each review by an ID number
df = df.reset_index().rename(columns = {'index': 'Id'})
```

In [ ]:
```
# Verify creating of indexed ID column
df
```

In [ ]:
```
import textblob
from textblob import Word

df['lem_review'] = df['stop_review'].apply(lambda x: " ".join(Word(word).lemmatize() for word i
```

In [ ]:
```
# Verify lemmatization has worked
df.head()
```

In [ ]:
```
# Identify vocabulary size
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['lem_review'])
print("Vocaulary size: ", len(tokenizer.word_index) + 1)
```

B1c. Proposed Word Embedding Length

In [ ]:
```
# Calculate proposed word embedding length
vocab_size = len(tokenizer.word_index) + 1
max_sequence_embedding = int(round(np.sqrt(np.sqrt(vocab_size)), 0))
print("The proposed word embedding length: ", max_sequence_embedding)
```

This is the result of taking the square of the square of our vocab size and will be the value used for our proposed embedding length. This was discussed in Dr. Elleh's lecture as well as the DataCamp materials.

B1d. Statistical Justification For The Chosen Maximum Sequence Length

```
In [ ]: # Create list with length of characters in each review
        commentary_length = []
        for char_len in df['lem_review']:
            commentary_length.append(len(char_len.split(' ')))

        # Calculate maximum, minimum, and median character length of each review
        commentary_max = np.max(commentary_length)
        commentary_min = np.min(commentary_length)
        commentary_median = np.median(commentary_length)
        print(" The maximum length of our sequences would be: ", commentary_max)
        print(" The minimum length of our sequences would be: ", commentary_min)
        print(" The median length of our sequences would be: ", commentary_median)
```

After creating a list of character length for each review I calculated the maximum, minimum, and median of each review. Based on this calculation our maximum sequence length will be set at 620. This will help us to evaluate our reviews by preserving input data so that the generated model doesn't yield conclusions that are likely not to generalize well. We will address shorter inputs through the use of padding below.

B2. Tokenization Process

The goals of the tokenization process is to separate the text into smaller chunks or 'tokens'. A token can include words, characters, or sub-words. We will assign "word_index" to each word in the dataset which will in turn help the model during the training process.

```
In [ ]: # Tokenize lem_review column
        def tokenize(text):
            tokens = re.split('\W+', text)
            return tokens

        # Put output into new column
        df['token_review'] = df['lem_review'].apply(lambda x: tokenize(x))
        df.head()
```

B3. Padding Process

Padding is a process used for neural networks to ensure that sequences have a consistent length. Because neural networks require inputs to have the same shape and size, this is a necessary step as our reviews are all different sizes. By performing padding on our dataset we will ensure that what we input into the neural network will be sentences of the same size. In this case I am using post padding as part of my preprocessing sequence.

```
In [ ]: # Define X and y variables
        X = df['token_review']
        y = df['sentiment']

        # Split the data into training and test sets
        X_train, X_data, y_train, y_data = train_test_split(X,y, test_size=0.3, random_state=42)

        # Split X_data and y_data in half to produce a validation set
        X_validation, X_test, y_validation, y_test = train_test_split(X_data,y_data, test_size=0.5, ranc

        # We now have 3 datasets to work with. 70% train, 15% test, and 15% validation
```

In [ ]:
```
# Validate size of each dataset
print(f"Training set size X:{X_train.shape}, y:{y_train.shape}")
print(f"Test set size X:{X_test.shape}, y:{y_test.shape}")
print(f"Validation set size X:{X_validation.shape}, y:{y_validation.shape}")
```

In [ ]:
```
# Modifying size of test set to match validation set
y_test = y_test[:-1]
X_test = X_test[:-1]
```

In [ ]:
```
# Validate size of each dataset
print(f"Training set size X:{X_train.shape}, y:{y_train.shape}")
print(f"Test set size X:{X_test.shape}, y:{y_test.shape}")
print(f"Validation set size X:{X_validation.shape}, y:{y_validation.shape}")
```

In [ ]:
```
# Initialize the tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

# Convert text to numerical sequences
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)
validation_sequences = tokenizer.texts_to_sequences(X_validation)

# Pad sequences to ensure consistent length
max_sequence_length = 620
X_train_padded = pad_sequences(train_sequences, maxlen=max_sequence_length, padding='post')
X_test_padded = pad_sequences(test_sequences, maxlen=max_sequence_length, padding='post')
X_valid_padded = pad_sequences(validation_sequences, maxlen=max_sequence_length, padding='post'
```

In [ ]:
```
print(X_train_padded.shape)
print(X_test_padded.shape)
print(X_valid_padded.shape)
```

In [ ]:
```
# Display the tain_data padded sequence
np.set_printoptions(threshold=sys.maxsize)
X_train_padded[1]
```

In [ ]:
```
# Display the test_data padded sequence
np.set_printoptions(threshold=sys.maxsize)
X_test_padded[1]
```

B4. Identify How Many Categories of Sentiment Will be Used

In this dataset the sentiment consists of a binary distribution. The sentiment is either positive (1) or negative (0).

B5. Steps For Data Preparation

1. Import necessary libraries and packages needed to use Tensorflow and NLTK operations.
2. Load 3 individual txt files for reviews from amazon, imdb, and yelp.
3. Concatenate all three sets of reviews into one pandas DataFrame
4. Detect and verify lack of null values within DataFrame.
5. Convert any non-string values to string values.
6. Review all unique characters in the dataset.
7. Remove accented letters from review column.
8. Remove numbers from review column.
9. Remove punctuation from review column.

10. Make all letters in review column lower case.
11. Verify absence of all special characters in review column.
12. Determine vocabulary size by doing the following:
    - Use list of stop words from nltk
    - Remove those words from reviews
    - Review most frequent words in the reviews
    - Determine a second list of stop words to also remove based on frequency of words that aren't needed to determine sentiment
    - Remove custom list of stop words
    - Verify words have been removed
13. Create a list with the length of each review and review the shortest and longest review
14. Visualize distribution of review lengths.
15. Review the length of the 15 longest reviews.
16. Create an index to provide an ID for each review in the DataFrame.
17. Lemmatize reviews and verify that it was succesful.
18. Use tokenizer on lemmatized reviews to determine vocabulary size.
19. Use the vocabulary size to calculate the word embedding length.
20. Use numpys to calculate the maximum, minimum, and median length of sequences.
21. Tokenize review column.
22. Split the dataset into train (70%), test (15%), and validation (15%) sets.
23. Verify the shape and modify the size of the test set to match the size of the validation set.
24. Convert the text to numerical sequences using texts_to_sequences
25. Pad the sequences using pad_sequences.
26. Display the padded sequences to verify that is was successful.

B6. Provide Copy of Pepared Dataset

In [ ]:
```
# Export current DataFrame to CSV
df.to_csv('cleaned_sentiment.csv')
```

C1. Model Summary

In [ ]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(4754, 256, input_length=max_sequence_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid'),
])

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

C2. Model Layers and Parameters

This is a sequential model that consists of 5 layers.

The first layer is an embedding layer. This allows us to feed it all of the data within the padded training set. It is able to do this by replacing our data with numbers, similar to the process that one-hot encoding uses. In this model, the layer is composed of 1,217,024 patameters as a result of the dimensions of the input data, the size of the vocabulary, and the output_dim being set at 256.

The second layer is flattening layer. This exists to convert all the input data into a single dimension. Basically, this layer helps to clean everything up and allow the following dense layers to do their thing.

The third layer is a dense layer consisting of 64 nodes, set to use the ReLu function. The purpose of this layer is to start narrowing down the input data from the flattening layer. This will take the output of the flattening layer which consists of 158,720 dimensions down to just 64. It is also able to handle 10,158,144 parameters. This number comes from the number of nodes, multiplied by the input from the previous layer, and another round of output.

The fourth layer is the second dense layer in the model, this one consisting of 32 nodes, and also set to use the ReLu function. This layer will futher narrow down the input data. This layer consists of only 2,080 parameters which is calculated like the previous dense layer.

The fifth and final dense layer consists of a singular node, this time using the sigmoid function. This layer will produce the final output of the model in a single value. This value will reflect the sentiment of the review whcih can be 1 (positive) or 0 (negative). This final layer has 33 parameters. Again, it's computed the same as the previous dense

C3. Justification of Hyperparameters

• activation functions

The activation functions chosen in my model were ReLu and sigmoid. The ReLu functions were chosen as those are considered industry standard due do its overall high performance. The sigmoid function was chosen due to it being binary. In a sigmoid function the minimum and maximum outcomes are 0 and 1. This is needed due to our sentiments being 0 or 1.

• number of nodes per layer

The number of nodes per layer has been thoroughly explained above. The embedding layer has 128 nodes, the flatten layer is just a conversion layer, the three dense layers that follow have nod evalues of 64, 32, and 1 respectively. All have the goal of narrowing down the data into a single output of 1 or 0.

• loss function

For my model I chose binary_crossentropy for my loss function. Because I am trying to solve a binary classification problem and not a regression problem this is the obvious loss function to choose. As said, the purpose of this analysis to classify reviews as either 1 (positive) or 0 (negative) sentiment. Because of this being a binary classfication analysis we should use the binary_crossentropy loss function.

• optimizer

The optimizer chosen for my model is "Adam". Just like the ReLu function, this optimizer is considered industry standard due to its high performance. Moving away from "Adam" would require justification, and in this analysis there is no justification to move away from it.

• stopping criteria

I used the val_accuracy as my stopping criteria. This allows the model to be ran efficiently by the whoever is using the model if the input data were to be modified.

• evaluation metric

To evaluate my model I will use the accuracy percentage from calling the model.evaluate function on our padded test data, and y_test data.

D1. Early Stopping Criteria on Model

```
In [ ]: # Define stopping criteria
        early_stopper = EarlyStopping(monitor='val_accuracy', patience=2)

        # Train the model
        results = model.fit(X_train_padded, y_train, validation_data = (X_valid_padded, y_validation),
                            epochs=10, callbacks=early_stopper)
```

As we can see the early stopper stopped the model from running after the sixth epoch. The val_accuracy improved drastically after the first epoch. The fifth and sixth epoch had the same val_accuracy and the early stopper kicked in as the patience was set to 2.

D2. Fitness of the Model

In looking at the accuracy plot we can see the training accuracy increase in each epoch while the validation accuracy plateaus at the fourth epoch (3 on the x-axis).

The beginning of the model overfitting can be observed in the loss plot as well. We can see the training loss decrease through each epoch while the validation loss starts increasing in the fourth epoch (3 on the x-axis). This trend indicates that the model is starting to overfit.

In an effort to address overfiting I used the early stopper based on val_accuracy. If we continued to let this model run through more epochs we would be in danger of overfit.

```
In [ ]: plt.plot(results.history['accuracy'], label= "Training Accuracy")
        plt.plot(results.history['val_accuracy'], label= "Validation Accuracy")
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.legend()
        plt.title("Accuracy of Training Model")
        plt.show()
```

```
In [ ]: plt.plot(results.history['loss'], label= "Training Loss")
        plt.plot(results.history['val_loss'], label= "Validation Loss")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.legend()
        plt.title("Loss of Training Model")
        plt.show()
```

D3. Model Evaluation Metric

```
In [ ]: # Evaluate the model on the test set
        test_loss, test_accuracy = model.evaluate(X_test_padded, y_test)

        # Print the test accuracy and test loss
        print("Test Accuracy:", round(test_accuracy*100, 1))
        print("Test Loss:", round(test_loss*100, 1))
```

In D2 you can see a line graph of the loss and here in D3 you can see the chosen evaluation metric, the accuracy and loss percentage.

Our chosen evaluation metric shows a loss of 53.7% and an accuracty of 80.3%. This is not a very strong model.

D4. Predictive Accuracy

We can use a confusion matrix to visualize the accuracy of our predictions.

In [ ]:
```
# Compile predictions for y from the model's X_test data
y_pred = model.predict(X_test_padded)
```

In [ ]:
```
# Round probabilities from the model
y_pred = np.rint(y_pred)
# Produce confusion matrix from y_pred variable and y_test
print("PREDICTED SENTIMENT")
print("  POS | NEG")
matrix_plot = confusion_matrix(y_test, y_pred)
print(f"  {matrix_plot[0]} POS")
print(f"  {matrix_plot[1]} NEG")
```

The confusion matrix shows a total of (183 + 148) 331 correct predictions versus (52 + 29) 81 incorrect predictions. The bottom row is all the negative sentiment and the top row is the positive sentiment. Overall, the model did a much better job of predicting positive sentiment from the reviews as there is a higher proportion of true positives in the top row than there are true negatives in the bottom row.

E. Saved Trained Network

Below is the code to save the trained network within the neural network.

In [ ]:
```
# Save model
# Create a file name, save it, and notify user if it was successful
file_name = "UCISentimentAnalysisModel.h5"
model.save(file_name)
print(f"Sentiment Analysis Model successfully saved as {file_name}")
```

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

This neural network is not very effective at predicting whether a user review is positive or negative. The calculated loss is 53.7%, and the calculate accuracy leaves a lot to be desired at 80.3%. Based on our course materials we want loss to be low and accuracy to be high. Our neural network is not very confident in its predictions as well as only being accuracte 8 out of 10 times. There are probably a few reasons for this.

The network architecture is basic. I tried to use LSTM layers, however the computing time was not cost-effective for me as a student. If I had access to cloud computing I am sure I could have developed a much more complex architecture that could have produced much more accuracte results with less loss.

I could have provided more layers for the model to work through. In this model we reduced our data over three dense layers from a very large input. Adding more dense layers could have helped produce a model that was more accuracte. However, again this was prohibitive due to the time it took for the model to iterate through each epoch.

Finally, the data cleaning could probably have been more effective. As part of my analysis I did discover that there were contradictory words in reviews. This is to say that words with positive connotations were in reviews with negative sentiment and vice-versa. I am sure I could have included that in my data cleaning, however, nothing about that was mentioned in the rubric or the lectures I had access to. Therefore, I skippe that step. I'm sure if I removed those reviews the neutral network would have been far more efficient and effective.

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

Based on my results my recommended course of action would be to put together a group of experts on neural networks and try to dig into the data and network architecture more. We should also be provided access to a cloud based computing service so we can built a more robust neural network and reduce our time spent running models.

Part VI: Reporting

H. 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.

The jupyter notebook is provided as a submission to this task in PDF form. It contains all relevant reports, code, output, and figures.

I. Denote specific web sources you used to acquire segments of third-party code that was used to support the application.

https://stackoverflow.com/questions/37926248/how-to-remove-accents-from-values-in-columns (https://stackoverflow.com/questions/37926248/how-to-remove-accents-from-values-in-columns) to catalog and remove accented figures in columns

Everything else was taken from DataCamp, Dr. Elleh's lectures, or previous submissions. I did have to manipulate Dr. Elleh's code and accessed that appropriate documentation for that.

J. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

No sources cited.