Advanced Data Analytics

D213 - Task 2

Jeremy Dorrough

Western Governors University

WGU Student ID# 000994113

A1

Utilizing neural network models and Natural Language Processing (NLP), can an examination of customer review wording be used to deduce their likes or dislikes, so that the organizations can improve their products and services?

A2

The objective of this analysis is to predict how customers feel about a product, allowing the organizations to improve products and address customer concerns. This means that goals will include using various Natural Language Processing (NLP) techniques for processing and transforming customer reviews before using a neural network model to accurately classify based on sentiment, either positive or negative.

A3

For this sentiment classification task, a TensorFlow-based neural network has been identified as an appropriate tool for text classification tasks, making accurate predictions using both the text and sentiment data (Yu & Malan, n.d.).

B1

a. The code processes the 'sentence' column of the concatenated_data dataframe to remove special characters such as emoji or non-English characters using a regular expression. After the removal, the sentences are converted to lowercase to ensure uniformity.
b. The vocabulary size was identified in the dataset. Sentences were converted to a list-of-lists, flattened, and counted. There were 4780 unique words in the dataset. The most frequently occurring words were examined, with "good" being the most common word, having repeated 178 times, followed by "movie" at 177 words. Additional results can be examined in the provided code.

c. The embedding dimension is set to 16, as evident from the EMBED_DIM = 16 statement. This means that each word in the vocabulary will be represented by a 16-dimensional vector. Setting the embedding dimension to 16 allows for computationally compact analysis while retaining detailed information.

d. The sentence column was tokenized in the training and test sets and the sequences were padded with a maximum length of 50. This lets the most important words train the model without it being overly influenced by less predictive data. Only five sequences exceeded the maximum length of 50.

B2

The goal of tokenization is to convert sentences into sequences of integers representing words. It helps in making the textual data understandable for the model. Note, the normalization had removed special characters and converted all text to lowercase. The code uses the Keras Tokenizer to process the sentences into sequences of integers. The Keras pad_sequences function is used to make sure that the tokenized sequences have the same specified maximum length.

B3

The padding standardizes sequence lengths so that text data is uniform for processing. The sequences are padded to a length of 50 using pad_sequences from TensorFlow after the text sequence, as indicated by 'post' in the padding parameter. If a sequence exceeds the set length, it's truncated. In the below screenshot is a single padded sequence example:

```
# Show single padded
print("Original:", train['sentence'].iloc[0])
print("Tokenized:", train_sequences[0])
print("Padded:", train_padded[0])
```

```
Original: way plug us unless go converter
Tokenized: [54, 220, 174, 437, 43, 1842]
Padded: [  54  220  174  437   43 1842    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
```

B4

The analysis evaluates customer sentiment, either favorable or unfavorable, which is represented in a binary format as either a 1 or 0, so there are two categories of sentiment: positive and negative. Considering this, the final layer will use a 'sigmoid' activation function to make the distinction between the two outcomes (Tuzsuz, n.d.).

B5

All necessary packages and libraries were imported for data manipulation, calculations, and visualizations. The data was imported and initially examined. It was provided in three datasets: Amazon, IMDB, and Yelp, which were concatenated into a single dataframe with labeling for the source of each review. The data was then checked for null values and examined.

The sentiment distribution was visualized with a histogram. Stopwords were removed from the sentence column, and the prepared data was exported. The data was divided into training and test sets with 80% (2198 values) used for training and 20% (550 values) for testing. The split was chosen to maximize the predictive accuracy of the model.

B6

See attached CSV.

C1

See the TensorFlow model summary below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 16)            76496

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 16)                272

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 76,785
Trainable params: 76,785
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
69/69 [==============================] - 1s 3ms/step - loss: 0.6923 - accuracy: 0.5173 - val_loss: 0.6957 - val_accuracy: 0.4527
Epoch 2/100
69/69 [==============================] - 0s 2ms/step - loss: 0.6890 - accuracy: 0.5182 - val_loss: 0.6922 - val_accuracy: 0.4527
Epoch 3/100
69/69 [==============================] - 0s 2ms/step - loss: 0.6809 - accuracy: 0.5623 - val_loss: 0.6856 - val_accuracy: 0.5000
Epoch 4/100
69/69 [==============================] - 0s 2ms/step - loss: 0.6598 - accuracy: 0.7475 - val_loss: 0.6682 - val_accuracy: 0.7255
Epoch 5/100
69/69 [==============================] - 0s 2ms/step - loss: 0.6170 - accuracy: 0.8367 - val_loss: 0.6446 - val_accuracy: 0.6745
Epoch 6/100
69/69 [==============================] - 0s 2ms/step - loss: 0.5512 - accuracy: 0.8822 - val_loss: 0.6124 - val_accuracy: 0.6964
Epoch 7/100
69/69 [==============================] - 0s 1ms/step - loss: 0.4709 - accuracy: 0.9117 - val_loss: 0.5511 - val_accuracy: 0.7873
Epoch 8/100
69/69 [==============================] - 0s 1ms/step - loss: 0.3952 - accuracy: 0.9190 - val_loss: 0.5170 - val_accuracy: 0.7836
Epoch 9/100
69/69 [==============================] - 0s 1ms/step - loss: 0.3265 - accuracy: 0.9404 - val_loss: 0.5130 - val_accuracy: 0.7618
Epoch 10/100
69/69 [==============================] - 0s 1ms/step - loss: 0.2768 - accuracy: 0.9422 - val_loss: 0.4787 - val_accuracy: 0.7873
Epoch 11/100
69/69 [==============================] - 0s 1ms/step - loss: 0.2358 - accuracy: 0.9504 - val_loss: 0.4713 - val_accuracy: 0.7836
Epoch 12/100
69/69 [==============================] - 0s 1ms/step - loss: 0.2062 - accuracy: 0.9550 - val_loss: 0.4955 - val_accuracy: 0.7600
Epoch 13/100
69/69 [==============================] - 0s 1ms/step - loss: 0.1802 - accuracy: 0.9604 - val_loss: 0.4617 - val_accuracy: 0.7909
Epoch 14/100
69/69 [==============================] - 0s 1ms/step - loss: 0.1600 - accuracy: 0.9636 - val_loss: 0.4733 - val_accuracy: 0.7745
Epoch 15/100
53/69 [======================>.......] - ETA: 0s - loss: 0.1401 - accuracy: 0.9670Restoring model weights from the end of the best epoch: 13.
69/69 [==============================] - 0s 1ms/step - loss: 0.1434 - accuracy: 0.9659 - val_loss: 0.4752 - val_accuracy: 0.7782
Epoch 15: early stopping
18/18 [==============================] - 0s 739us/step - loss: 0.4617 - accuracy: 0.7909
Test Accuracy: 79.09%
```

C2

As observed in the provided screenshot, the model used has four layers: Embedding, GlobalAveragePooling1D layer, and two Dense layers. In total, the model has 76,785 trainable parameters. The Embedding layer converts input sequence of word indices to dense vectors of fixed size of 16 and has 76,496 parameters. The GlobalAveragePooling1D layer averages the feature dimensions, turning the 2D tensors into 1D tensors and has 0 parameters. The first fully connected Dense Layer has 6 nodes and uses 272 parameters while the second Dense layer has a single node with 17 parameters.

C3

a. The 'ReLU' activation function is used in the hidden layer for its non-linearity and efficiency.

b. For binary classification, the 'sigmoid' activation function is adopted in the output layer, ensuring outputs between 0 and 1.

c. The embedding layer uses a 16-dimensional vector for each word, followed by a dense layer with 16 nodes to capture feature relationships; the final layer has one node for binary classification.

d. The 'binary_crossentropy' loss function was used to measure the difference between true and predicted labels.

e. The 'adam' optimizer is employed because it is lightweight and efficient, with limited need for manual tuning.

f. For stopping criteria, to mitigate overfitting, early stopping is applied, monitoring 'val_loss'. If there's no improvement over two epochs, training stops and optimizes for the best weights.

g. Accuracy is used as the evaluation metric to confirm the model's ability to make accurate predictions.

D1

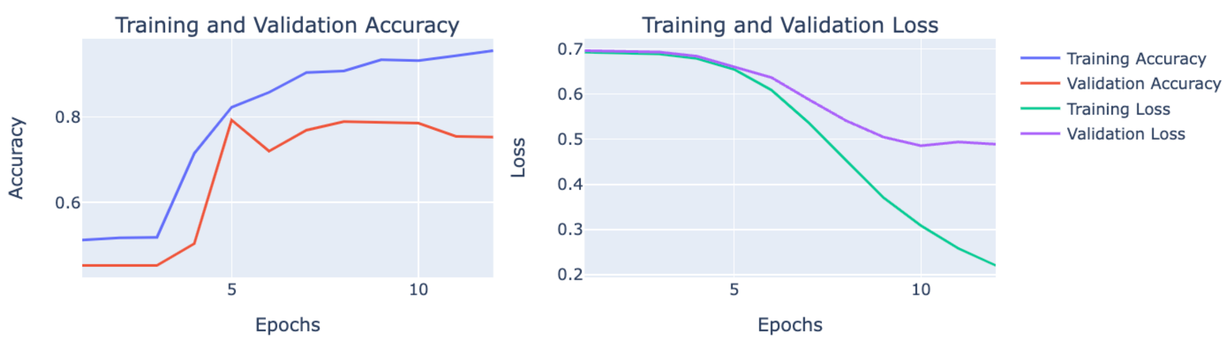
Using early stopping instead of a fixed epoch number addressed overfitting by halting training when the validation metric stabilizes, the result being that there is more efficient training and fewer unnecessary epochs. One potential issue with this is that it requires additional tuning and can potentially stop prematurely. The final epoch and it's stopping procedure can be examined in the screenshot below:

```
Epoch 15/100
53/69 [=====================>.......] - ETA: 0s - loss: 0.1401 - accuracy: 0.9670Restoring model weights from the end of the best epoch: 13.
69/69 [=============================] - 0s 1ms/step - loss: 0.1434 - accuracy: 0.9659 - val_loss: 0.4752 - val_accuracy: 0.7782
Epoch 15: early stopping
18/18 [=============================] - 0s 739us/step - loss: 0.4617 - accuracy: 0.7909
Test Accuracy: 79.09%
```

D2

The model has an embedding layer followed by global average pooling and two dense layers, as discussed and has an accuracy of 79.09%. To address overfitting, early stopping was implemented, and validation loss was monitored. The early stopped halted training if the validation loss did not improve for two consecutive epochs, preventing overfitting or added computational work. The early stopping also can revert to the best weights, using the epoch with the best performance and the model seems reasonably fit.

The visualization (below) also indicates good fit; training and validation accuracy increase while training and validation loss decrease, indicating increasing accuracy. Also, training accuracy is higher than validation, so the model doesn't appear to be overfitting.



D3

The test accuracy of the trained model is 79.09%. The trained network was applied on individual test sentences, as demonstrated in the code, it shows that it correctly classifies most of the sentences but also makes several misclassifications.

Precision for negative sentiments (0) is 0.84, meaning 84% of the samples predicted correctly and recall is .77 (77%) for actual negative samples. For positive sentiments (1), precision is 0.68, and recall is 0.88. The confusion matrix shows 231 true negatives along with 45 false negatives, 70 false positives with 204 true positives.

The model appears to ave decent predictive accuracy for the test set but could be tuned further, especially for correcting the high number of false positives and negatives. To summarize, it seems that the code is reasonably accurate but could be enhanced.

F

This network is designed to be both simple and efficient, while remaining accurate in prediction for binary questions. With minimal layers, it's ideal for straightforward training where there are minimal computational resources. The inclusion of embedding and pooling layers means the model can understand longer texts in their entirety. Instead of focusing on specific, individual words, it emphasizes the overall sentiment or themes in the text.

That being said, the network utilizes a comparatively generous number of nodes. This allows it to identify non-specified patterns in the new data. The network is designed to address binary questions, like the positive/negative question in the sentiment data. The network will efficiently analyze the text to make a binary determination.

H

See attached.

G

Given the achieved test accuracy of 79.09% and the decreasing gap between training and validation accuracy, it suggests relatively good fitting. The limitations of the analysis should be taken into consideration and further data gathering or model tuning should be performed. That said, this model can be used in an organizational setting to make general predictions where a moderately accurate prediction is appropriate.

I

w3resource. (2022, August 19). Python Regular Expression. https://www.w3resource.com/python/python-regular-expression.php

GeeksforGeeks. (2023, May 16). Removing stop words with NLTK in Python. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

TensorFlow. (2023, May 27). Word embeddings. https://www.tensorflow.org/text/guide/word_embeddings

Brownlee, J. (2016, June 17). Display Deep Learning Model Training History in Keras. Machine Learning Mastery. https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

Safa Mulani. (2020, October 13). Python predict() function - All you need to know! AskPython. https://www.askpython.com/python/examples/python-predict-function

Amaratunga, T. (2020, October 12). Fixing the KeyError: ‘acc’ and KeyError: ‘val_acc’ Errors in Keras 2.3.x or Newer. Towards Data Science. https://towardsdatascience.com/fixing-the-keyerror-acc-and-keyerror-val-acc-errors-in-keras-2-3-x-or-newer-b29b52609af9

J

Yu, B., & Malan, D. J. (n.d.). CS50’s Introduction to Artificial Intelligence with Python. https://cs50.harvard.edu/ai/2020/notes/5/

Tuzsuz, D. (n.d.). Sigmoid Function. LearnDataSci. https://www.learndatasci.com/glossary/sigmoid-function/

In [1]:
```python
import pandas as pd
import zipfile
import io

import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

import plotly.graph_objects as go
from plotly.subplots import make_subplots

from collections import Counter

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model

from sklearn.metrics import classification_report, confusion_matrix
```

```
2023-09-18 16:01:40.416360: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to us
e available CPU instructions in performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2, in other operations, rebuild TensorFlow with the appropriate compiler
flags.
```

In [2]:
```python
# Import
zip_path = "sentiment_labelled_sentences.zip"

zipped_txts = {
    "sentiment labelled sentences/amazon_cells_labelled.txt": "amzn",
    "sentiment labelled sentences/imdb_labelled.txt": "imdb",
    "sentiment labelled sentences/yelp_labelled.txt": "yelp"
}

# Dictionary to hold dataframes
dfs = {}

with zipfile.ZipFile(zip_path, 'r') as z:
    for file_path, df_name in zipped_txts.items():
        with z.open(file_path) as f:
            dfs[df_name] = pd.read_csv(io.TextIOWrapper(f), sep="\t", header=None, names=["sentence", "label"])

print(dfs['amzn'].head())
print(dfs['imdb'].head())
print(dfs['yelp'].head())
```

```
                                            sentence  label
0  So there is no way for me to plug it in here i...      0
1                        Good case, Excellent value.      1
2                             Great for the jawbone.      1
3  Tied to charger for conversations lasting more...      0
4                                  The mic is great.      1
                                            sentence  label
0  A very, very, very slow-moving, aimless movie ...      0
1  Not sure who was more lost - the flat characte...      0
2  Attempting artiness with black & white and cle...      0
3       Very little music or anything to speak of.      0
4  The best scene in the movie was when Gerardo i...      1
                                            sentence  label
0                           Wow... Loved this place.      1
1                                 Crust is not good.      0
2        Not tasty and the texture was just nasty.      0
3  Stopped by during the late May bank holiday of...      1
4  The selection on the menu was great and so wer...      1
```

In [3]:
```python
# Map source to value
source_mapping = {
    "amzn": 1,
    "imdb": 2,
    "yelp": 3
}

# List to insert source value
dfs_list = []

# Insert source value
for df_name, source_value in source_mapping.items():
    df = dfs[df_name].copy()
    df["source"] = source_value
```

```python
    dfs_list.append(df)

# Concatenate into one combined df
concatenated_data = pd.concat(dfs_list, ignore_index=True)

# Verify
print(concatenated_data.head())
```

```
                                            sentence  label  source
0  So there is no way for me to plug it in here i...      0       1
1                        Good case, Excellent value.      1       1
2                             Great for the jawbone.      1       1
3  Tied to charger for conversations lasting more...      0       1
4                                  The mic is great.      1       1
```

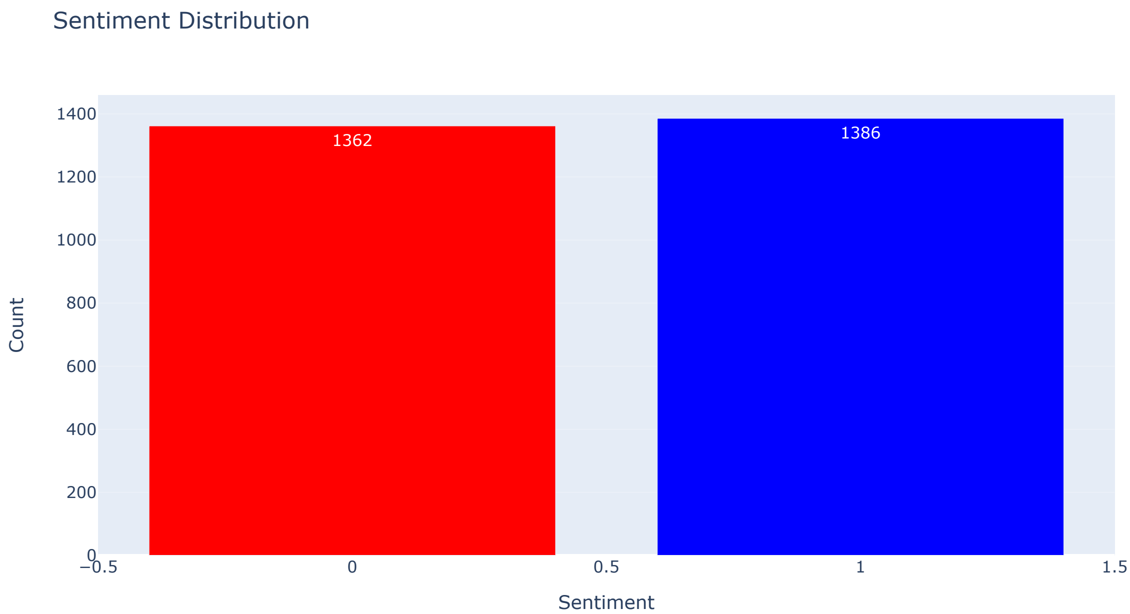
In [4]:
```python
# Check for nulls
concatenated_data.isna().any()
```

Out[4]:
```
sentence    False
label       False
source      False
dtype: bool
```

In [5]:
```python
# View unmodified sentences
print("Before removing special characters and casing:\n")
print(concatenated_data['sentence'].head())


# Strip special characters (w3resource, 2022)
concatenated_data['sentence'] = concatenated_data['sentence'].str.replace(r'[^a-zA-Z0-9\s]', '', regex=True)

# Change to lowercase
concatenated_data['sentence'] = concatenated_data['sentence'].str.lower()

# Verify
print("\nAfter removing special characters and casing:\n")
print(concatenated_data['sentence'].head())
```

```
Before removing special characters and casing:

0    So there is no way for me to plug it in here i...
1                          Good case, Excellent value.
2                               Great for the jawbone.
3    Tied to charger for conversations lasting more...
4                                    The mic is great.
Name: sentence, dtype: object

After removing special characters and casing:

0    so there is no way for me to plug it in here i...
1                             good case excellent value
2                                 great for the jawbone
3    tied to charger for conversations lasting more...
4                                       the mic is great
Name: sentence, dtype: object
```

In [6]:
```python
# Get counts
label_counts = concatenated_data['label'].value_counts()

# Colors
colors = ['blue', 'red']

# Plot
fig = go.Figure(data=[
    go.Bar(x=label_counts.index, y=label_counts.values, text=label_counts.values, textposition='auto', marker_color=colors
])

# Label
fig.update_layout(
    title='Sentiment Distribution',
    xaxis_title='Sentiment',
    yaxis_title='Count'
)

# Show
fig.show()
```



In [7]:
```
# Drop stopwords (GeeksforGeeks, 2023)
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
concatenated_data['sentence'] = concatenated_data['sentence'].apply(lambda x: ' '.join([word for word in x.split() if word

print(concatenated_data.head())
```

```
                                          sentence  label  source
0                    way plug us unless go converter      0       1
1                          good case excellent value      1       1
2                                      great jawbone      1       1
3  tied charger conversations lasting 45 minutesm...      0       1
4                                          mic great      1       1
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jeremydorrough/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [8]:
```
# Export CSV
csv_path = "prepared_data.csv"
concatenated_data.to_csv(csv_path, index=False)
```

In [9]:
```
# Train/test split
split = int(len(concatenated_data) * .8)
train = concatenated_data[:split]
test = concatenated_data[split:]
print('Training:', len(train), 'values')
print('Test:', len(test), 'values')
```

```
Training: 2198 values
Test: 550 values
```

In [10]:
```
# Convert to list-of-lists, flatten, count, and verify
word_lists = train['sentence'].str.split().tolist()
all_words = [word for sublist in word_lists for word in sublist]
unique_words = set(all_words)
num_unique_words = len(unique_words)
print(f"Unique word count: {num_unique_words}")
```

```
Unique word count: 4780
```

In [11]:
```
# Get frequency and show top words
word_freq = Counter(all_words)
top_words = word_freq.most_common(100)

print("Top frequent words:")
for word, count in top_words:
    print(f"{word}: {count}")
```

```
Top frequent words:
good: 178
movie: 177
great: 169
phone: 162
film: 155
0: 138
one: 128
1: 126
like: 104
time: 90
bad: 88
really: 83
well: 77
dont: 70
would: 66
service: 63
best: 62
even: 62
ever: 62
quality: 59
also: 57
place: 56
product: 56
food: 56
works: 52
ive: 52
work: 52
made: 51
love: 49
sound: 48
use: 48
excellent: 47
headset: 47
could: 46
battery: 45
better: 45
never: 44
recommend: 43
im: 42
get: 41
back: 41
acting: 41
go: 40
see: 40
much: 38
make: 37
first: 37
didnt: 36
think: 36
characters: 36
ear: 35
nice: 35
still: 35
way: 34
worst: 32
case: 31
waste: 31
little: 31
2: 31
every: 31
right: 30
people: 30
pretty: 30
everything: 30
price: 30
got: 30
real: 30
movies: 30
enough: 29
look: 29
two: 28
doesnt: 28
thing: 28
story: 28
money: 27
plot: 27
films: 27
know: 26
seen: 26
say: 25
new: 25
disappointed: 25
piece: 25
```

```
used: 25
10: 25
cant: 25
worth: 24
terrible: 24
many: 24
nothing: 24
script: 24
wonderful: 24
far: 23
years: 23
life: 23
poor: 23
definitely: 23
going: 22
minutes: 22
anyone: 22
```

In [12]:
```python
# (TensorFlow, 2023)

# Tokenize
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train['sentence'])

train_sequences = tokenizer.texts_to_sequences(train['sentence'])
test_sequences = tokenizer.texts_to_sequences(test['sentence'])

# Pad
maxlen = 50
train_padded = pad_sequences(train_sequences, maxlen=maxlen, padding='post', truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=maxlen, padding='post', truncating='post')

# Verify
long_train_sequences = sum([1 for seq in train_sequences if len(seq) > maxlen])
long_test_sequences = sum([1 for seq in test_sequences if len(seq) > maxlen])

print(f"Number of sequences in train data exceeding maxlen: {long_train_sequences}")
print(f"Number of sequences in test data exceeding maxlen: {long_test_sequences}")
```

```
Number of sequences in train data exceeding maxlen: 5
Number of sequences in test data exceeding maxlen: 0
```

In [21]:
```python
# Show single padded
print("Original:", train['sentence'].iloc[0])
print("Tokenized:", train_sequences[0])
print("Padded:", train_padded[0])
```

```
Original: way plug us unless go converter
Tokenized: [54, 220, 174, 437, 43, 1842]
Padded: [  54  220  174  437   43 1842    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
```

In [14]:
```python
# (Brownlee, 2016)

# Parameters
VOCAB_SIZE = len(tokenizer.word_index) + 1
EMBED_DIM = 16

# Define
model = Sequential([
    Embedding(VOCAB_SIZE, EMBED_DIM, input_length=maxlen),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

# Summary
model.summary()

# Define early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=2, verbose=1, restore_best_weights=True)

# Train with early stopping
history = model.fit(
    train_padded,
    train['label'].values,
    epochs=100,
    validation_data=(test_padded, test['label'].values),
    verbose=1,
    callbacks=[early_stopping]
```

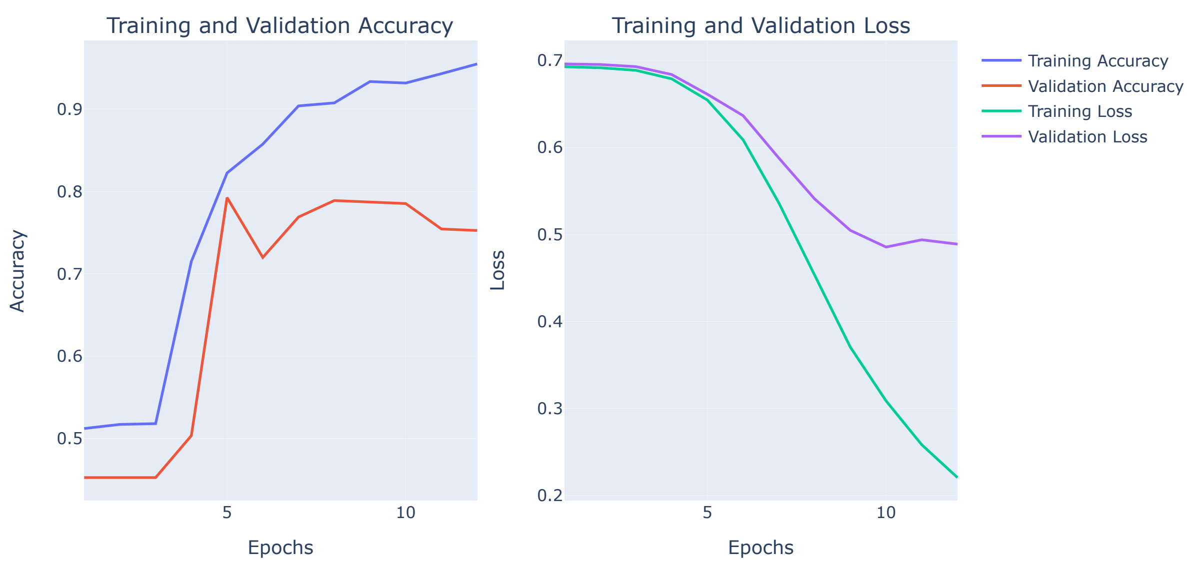
```python
)

# Evaluate
loss, accuracy = model.evaluate(test_padded, test['label'].values, verbose=1)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 50, 16)            76496     
                                                                 
 global_average_pooling1d (G  (None, 16)               0         
 lobalAveragePooling1D)                                          
                                                                 
 dense (Dense)               (None, 16)                272       
                                                                 
 dense_1 (Dense)             (None, 1)                 17        
                                                                 
=================================================================
Total params: 76,785
Trainable params: 76,785
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
69/69 [==============================] - 1s 3ms/step - loss: 0.6928 - accuracy: 0.5123 - val_loss: 0.6960 - val_accuracy: 0.4527
Epoch 2/100
69/69 [==============================] - 0s 1ms/step - loss: 0.6916 - accuracy: 0.5173 - val_loss: 0.6953 - val_accuracy: 0.4527
Epoch 3/100
69/69 [==============================] - 0s 1ms/step - loss: 0.6887 - accuracy: 0.5182 - val_loss: 0.6929 - val_accuracy: 0.4527
Epoch 4/100
69/69 [==============================] - 0s 1ms/step - loss: 0.6789 - accuracy: 0.7152 - val_loss: 0.6838 - val_accuracy: 0.5036
Epoch 5/100
69/69 [==============================] - 0s 1ms/step - loss: 0.6545 - accuracy: 0.8226 - val_loss: 0.6611 - val_accuracy: 0.7927
Epoch 6/100
69/69 [==============================] - 0s 1ms/step - loss: 0.6087 - accuracy: 0.8576 - val_loss: 0.6365 - val_accuracy: 0.7200
Epoch 7/100
69/69 [==============================] - 0s 1ms/step - loss: 0.5362 - accuracy: 0.9040 - val_loss: 0.5878 - val_accuracy: 0.7691
Epoch 8/100
69/69 [==============================] - 0s 1ms/step - loss: 0.4532 - accuracy: 0.9076 - val_loss: 0.5409 - val_accuracy: 0.7891
Epoch 9/100
69/69 [==============================] - 0s 1ms/step - loss: 0.3705 - accuracy: 0.9336 - val_loss: 0.5047 - val_accuracy: 0.7873
Epoch 10/100
69/69 [==============================] - 0s 1ms/step - loss: 0.3087 - accuracy: 0.9318 - val_loss: 0.4855 - val_accuracy: 0.7855
Epoch 11/100
69/69 [==============================] - 0s 2ms/step - loss: 0.2582 - accuracy: 0.9431 - val_loss: 0.4939 - val_accuracy: 0.7545
Epoch 12/100
49/69 [====================>.........] - ETA: 0s - loss: 0.2240 - accuracy: 0.9541Restoring model weights from the end of the best epoch: 10.
69/69 [==============================] - 0s 1ms/step - loss: 0.2205 - accuracy: 0.9550 - val_loss: 0.4889 - val_accuracy: 0.7527
Epoch 12: early stopping
18/18 [==============================] - 0s 725us/step - loss: 0.4855 - accuracy: 0.7855
Test Accuracy: 78.55%
```

In [15]:

```python
# (Amaratunga, 2020)
# Get training values
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Subplot
fig = make_subplots(rows=1, cols=2,
                    subplot_titles=('Training and Validation Accuracy', 'Training and Validation Loss'))

# Accuracy plots
fig.add_trace(go.Scatter(x=list(epochs), y=acc, mode='lines', name='Training Accuracy'), row=1, col=1)
fig.add_trace(go.Scatter(x=list(epochs), y=val_acc, mode='lines', name='Validation Accuracy'), row=1, col=1)

# Loss plots
fig.add_trace(go.Scatter(x=list(epochs), y=loss, mode='lines', name='Training Loss'), row=1, col=2)
fig.add_trace(go.Scatter(x=list(epochs), y=val_loss, mode='lines', name='Validation Loss'), row=1, col=2)
```

```python
# Formatting
fig.update_xaxes(title_text='Epochs', row=1, col=1)
fig.update_yaxes(title_text='Accuracy', row=1, col=1)
fig.update_xaxes(title_text='Epochs', row=1, col=2)
fig.update_yaxes(title_text='Loss', row=1, col=2)

# Show
fig.show()
```



In [16]:
```python
#(Mulani, 2020)
# Predict on Test Data
predictions = model.predict(test_padded)

# Classify Predictions
predicted_labels = [1 if p > 0.5 else 0 for p in predictions]

# Preview
for sentence, label, predicted_label in zip(test['sentence'].head(10), test['label'].head(10), predicted_labels[:10]):
    print(f"Sentence: {sentence}")
    print(f"Actual Label: {label}")
    print(f"Predicted Label: {predicted_label}")
    print("------\n")

# Analyze Predictions

print(classification_report(test['label'], predicted_labels))
print(confusion_matrix(test['label'], predicted_labels))
```

```
18/18 [==============================] - 0s 540us/step
Sentence: dont think well going back anytime soon
Actual Label: 0
Predicted Label: 0
------

Sentence: food gooodd
Actual Label: 1
Predicted Label: 0
------

Sentence: far sushi connoisseur definitely tell difference good food bad food certainly bad food
Actual Label: 0
Predicted Label: 1
------

Sentence: insulted
Actual Label: 0
Predicted Label: 0
------

Sentence: last 3 times lunch bad
Actual Label: 0
Predicted Label: 0
------

Sentence: chicken wings contained driest chicken meat ever eaten
Actual Label: 0
Predicted Label: 0
------

Sentence: food good enjoyed every mouthful enjoyable relaxed venue couples small family groups etc
Actual Label: 1
Predicted Label: 1
------

Sentence: nargile think great
Actual Label: 1
Predicted Label: 1
------

Sentence: best tater tots southwest
Actual Label: 1
Predicted Label: 1
------

Sentence: loved place
Actual Label: 1
Predicted Label: 1
------

              precision    recall  f1-score   support

           0       0.83      0.77      0.80       301
           1       0.74      0.81      0.77       249

    accuracy                           0.79       550
   macro avg       0.78      0.79      0.78       550
weighted avg       0.79      0.79      0.79       550

[[231  70]
 [ 48 201]]
```

In [17]:
```python
# Save model
model.save('sentiment_analysis_model.h5')
```

In [18]:
```python
# Load model
load = load_model('sentiment_analysis_model.h5')
```