WGU D213 PA 2

Andrew Shrestha

Part I: Research Question

A1. Question: Utilizing the NLP (Natural Language Processing) algorithm, will it be viable to identify whether future customer reviews will contain either a positive or negative outlook? This will be based on existing labelled data sets in combination with our NLP analysis.

A2. Objectives and Goals: The goal of this research analysis is to add a greater insight into predicting their overall outlook via customer feedback as well as which products our customers prefer. This knowledge will be very significant towards stakeholders as they will not only be able to more efficiently allocate their resources into products that their customers will prefer, but to also understand customer sentiments towards their products. These goals will be able to be achieved through the use of creating an NLP model using the TensorFlow opensource software. f

A3. Identification of Neural Network: The Neural Network being used in this report will be both TensorFlow and Keras. We will be using these two libraries in tandem to forecast either positive or negative customer outlook using our database consisting of written reviews and viewed items. After putting the NLP and above-mentioned neural networks through training, we will be able to provide the customer outlook through classification of either positive or negative.

Part II: Method Justification

B1. Exploration of Data

```
# Import standard libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from tensorflow import keras
from dateutil.parser import parse

import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
from matplotlib.axes._axes import _log as matplotlib_axes_logger
import nltk
import tensorflow as tf
nltk.download('punkt')
from nltk import word_tokenize
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
# Load data sets
amazon_df = pd.read_csv(r"C:\Users\andre\OneDrive\Desktop\amazon_label.txt",
names = ['review', 'outlook'], sep = '\t')
imdb_df = pd.read_csv(r"C:\Users\andre\OneDrive\Desktop\IMDB_label.txt",
names = ['review', 'outlook'], sep = '\t')
yelp_df = pd.read_csv(r"C:\Users\andre\OneDrive\Desktop\YELP_label.txt",
names = ['review', 'outlook'], sep = '\t')


# Take a look at the individual data sets before we choose to combine them
into a single data set by observing the top 5 rows
print(amazon_df.head(5))
```

```
                                              review  outlook
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
```

```
print(imdb_df.head(5))
```

```
                                              review  outlook
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
```

```
print(yelp_df.head(5))
```

```
                                              review  outlook
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
```

```
#Merge the data sets together

data_frames = [amazon_df, imdb_df, yelp_df]
from functools import reduce

df = reduce(lambda  left,right: pd.merge(left,right, on = ['review',
'outlook'], how = 'outer'), data_frames)


#Make sure that the data sets have been merged successfully
df
```

review	outlook	
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
...	...	...
2743	I think food should have flavor and texture an...	0
2744	Appetite instantly gone.	0
2745	Overall I was not impressed and would not go b...	0
2746	The whole experience was underwhelming, and I ...	0
2747	Then, as if I hadn't wasted enough of my life ...	0

2748 rows × 2 columns

```
#Proceed to do some exploritoty analysis on the data to get a better
understanding
df.info
```

```
<bound method DataFrame.info of
review  outlook
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
...                                                ...       ...
2743  I think food should have flavor and texture an...        0
2744                          Appetite instantly gone.         0
2745  Overall I was not impressed and would not go b...        0
2746  The whole experience was underwhelming, and I ...        0
2747  Then, as if I hadn't wasted enough of my life ...        0

[2748 rows x 2 columns]>
```

```
df.dtypes
```

```
review     object
outlook     int64
dtype: object
```

```
df.describe
```

```
<bound method NDFrame.describe of
review  outlook
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
...                                                ...       ...
2743  I think food should have flavor and texture an...        0
2744                          Appetite instantly gone.         0
2745  Overall I was not impressed and would not go b...        0
2746  The whole experience was underwhelming, and I ...        0
2747  Then, as if I hadn't wasted enough of my life ...        0

[2748 rows x 2 columns]>
```

```
# An important part of the data prepartion process is to make sure there are
no missing or null values in the data set
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
review     0
outlook    0
dtype: int64
```

```
#Take a look at the percentage of positive vs negative outlook according to
the reviews, where 1 = Positive and 0 = Negative
print (df.outlook.value_counts() / len(df))
```

```
1    0.504367
0    0.495633
Name: outlook, dtype: float64
```

```
#observe the shortest and longest reviews as well as total reviews
length_reviews = df.review.str.len()
length_reviews
```

```
0        82
1        27
2        22
3        79
4        17
       ...
2743     66
2744     24
2745     50
2746     91
2747    134
Name: review, Length: 2748, dtype: int64
```

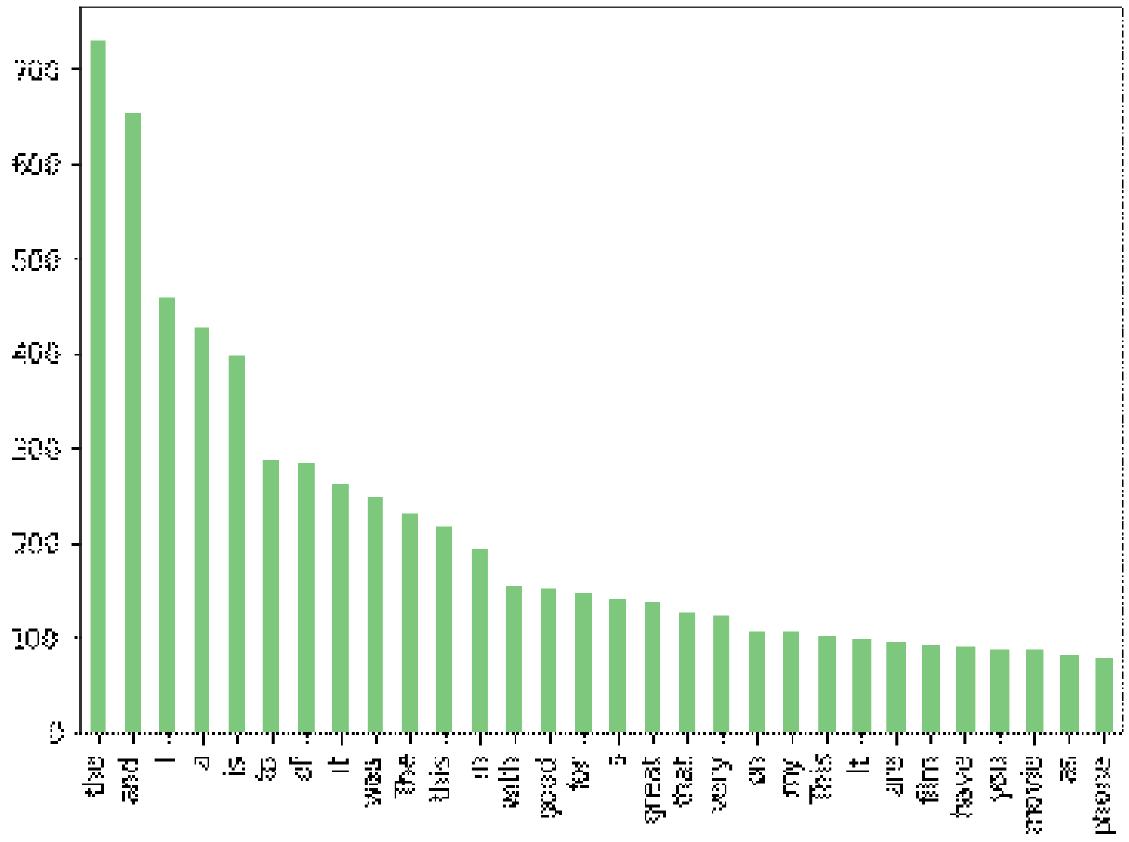
```
min(length_reviews)
7
```

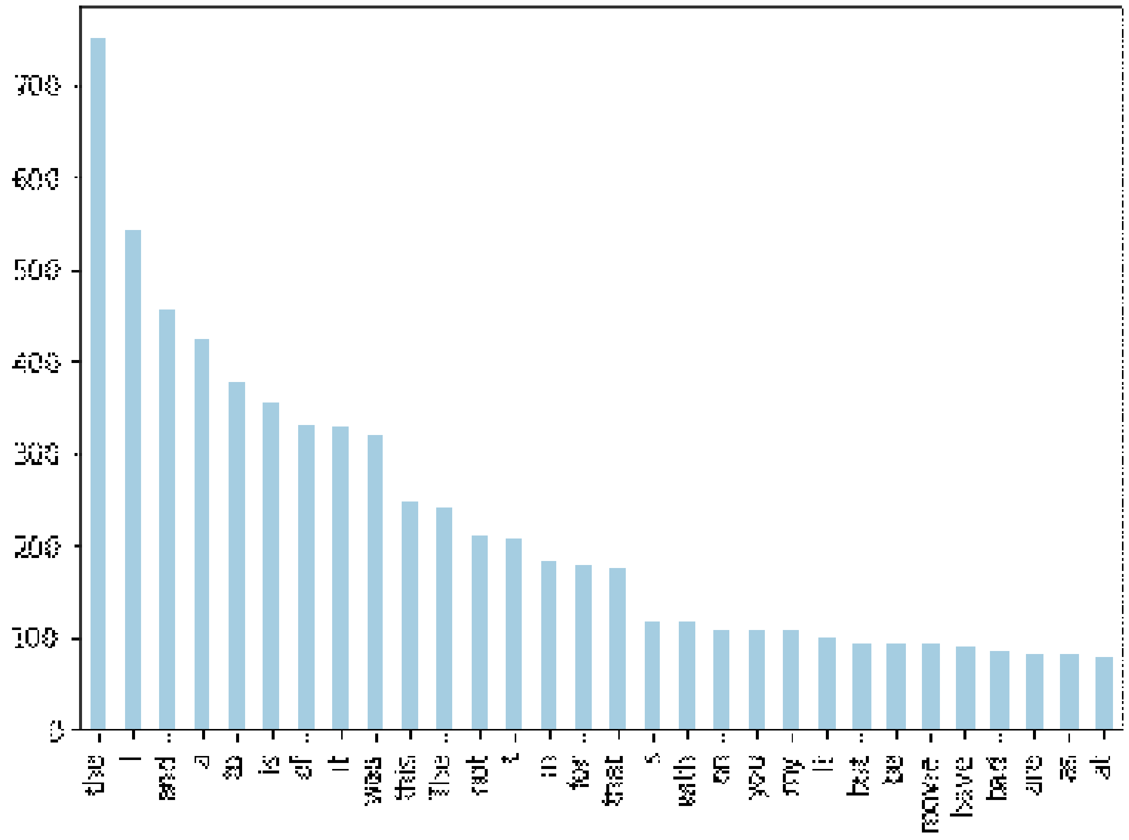
```
print( str(max(length_reviews)) )
8041
```

```
sum(length_reviews)
196812
```

```
# We want to go ahead and break up our strings into elements as tokens for
text mining
word_tokens = [word_tokenize(review) for review in df.review]
type(word_tokens)
list


#check for presence of unusual characters/emojis
for i in df.review:
    substring = ":)"
    if substring in i:
        print("smiley emoji")
smiley emoji
smiley emoji
smiley emoji
smiley emoji


# We have a function that gives us the number of words in each review. This
will help us with vocabulary size
def count_words(review):

    words = review.split()

    return len(words)

df['word_count'] = df['review'].apply(count_words)


#Find the vocabulary size of unique words in the dataframe
unique = set(df['review'].str.replace('[^a-zA-Z]',
'').str.lower().str.split(' ').sum())
print(str(len(unique)) + ' Vocab size unique words')

2717 Vocab size unique words

#clean up URLs and punctuation of text
import re
def clean_url(review_text):
    return re.sub('r.http\S+', '', review_text)


df['clean_review'] = df['review'].apply(clean_url)

def clean(txt):
    txt = txt.str.replace('(<br/>)', '')
    txt = txt.str.replace('(<a).*(>).*(</a>)', '')
    txt = txt.str.replace('(&amp)', '')
    txt = txt.str.replace('(&gt)', '')
    txt = txt.str.replace('(&lt)', '')
    txt = txt.str.replace('(\xa0)', '')
    return txt
```

```
# remove all miscellaneous characters
def clean_non_alphanumeric(review_text):
    return re.sub('[^a-zA-Z]', ' ', review_text)

# Refresh clean_review column
df['clean_review'] = df['clean_review'].apply(clean_non_alphanumeric)


# Remove infrequent word text
freq = pd.Series(' '.join(df['clean_review']).split()).value_counts()
less_freq = list(freq[freq == 1].index)


df['clean_review'] = df['clean_review'].apply(lambda x: " ".join(x for x in
x.split() if x not in less_freq))


# remove duplicated punctuation
df['clean_review'] = df['clean_review'].str.replace('[^\w\s]', '')


# Prepare our newly cleaned dataframe
negative_review = df[df.outlook == 0]['clean_review']
positive_review = df[df.outlook == 1]['clean_review']


color = ['Accent', 'Paired']
split_df = [positive_review, negative_review]


for item in range(3):
    try:
        plt.figure(figsize = (7, 5))
        pd.Series(' '.join([i for i in
split_df[item]]).split()).value_counts().head(30).plot(kind = 'bar', colormap
= color[item])
        plt.show();
    except IndexError:
        pass
```


0
100
200
300
400
500
600
700
the
and
I
a
is
to
of
it
was
The
this
in
with
good
for
s
great
that
very
on
my
This
It
are
film
have
you
movie
as
phone


0
100
200
300
400
500
600
700
the
I
and
a
to
is
of
it
was
this
The
not
t
in
for
that
s
with
on
you
my
It
but
be
movie
have
bad
are
as
at

```
# Now we remove the words present in both positive and negative reviews for
more accuracy and less ambiguity
def word_remover(review):
    return ' '.join([i for i in review.split() if i not in ['film', 'get',
'good', 'like', 'movi', 'phone', 'work']])

negative_review = negative_review.apply(word_remover)
positive_review = positive_review.apply(word_remover)
```

B2. Goals of Tokenization with Codes

Our goal for the tokenization is to break up our raw data code into individual tokens and then utilizing them in combinations the programs grammar logic to work with natural language processing. This breaking up text into fragments, we are able to apply rules to create larger meanings.

In our case, we obtain individual words from our reviews to obtain root meaning while at the same time removing insignificant verbiage. We can then fulfill the goal of translating text into encoded integers **(Tal Perry, 2021).**

```
# We now want to split our data into traning and testing sets, then using a
70% training and 30% testing ratio


X = df['clean_review']
y = df['outlook']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state = 42)


# We go ahead and tokenize our reviews

tokenizer = Tokenizer(oov_token = '<OOV>')

split = round(len(df) * 0.8)
training_reviews = df['clean_review'][:split]
training_outlook = df['outlook'][:split]
test_reviews = df['clean_review'][split:]
test_outlook = df['outlook'][split:]
```

```
training_sentence = []
training_label = []
test_sentence = []
test_label = []


for row in training_reviews:
    training_sentence.append(str(row))
for row in training_outlook:
    training_label.append(row)
for row in test_reviews:
    test_sentence.append(str(row))
for row in test_outlook:
    test_label.append(row)


# create our limitations for the vocabulary
vocab_size = 2000
embedding_dim = 16
max_length = 100
trunc_type = 'post'
oov_tok = '<OOV>'
padding_type = 'post'


tokenizer = Tokenizer(num_words = vocab_size, oov_token = oov_tok)


tokenizer.fit_on_texts(training_sentence)
word_index = tokenizer.word_index


print(str(len(word_index)) + ' total vocab words.')
```

```
2108 total vocab words.
```

B3. Padding Process used to Standardize

The process of Padding in Keras is in order to ensure that all of the inputted data will have the same length. To address our reviews with differing lengths, we either pad or truncate the input data, and thus they become standardized for our deep learning model.

By setting our Padding to equal ‘pre’ or ‘post’ will add padded place holders to either the start of the index or the end of the index to reach our chosen length of sequence. Similarly to this, we can set our truncating to equal ‘pre’ or ‘post’ to remove large values from the beginning or end respectively **(Data Flair, 2022).**

In our case for the reviews, we will be using the standardized length of sequence to be that of 100 words.

```
# We now create padding for our reviews
sequence = tokenizer.texts_to_sequences(training_sentence)
padded = pad_sequences(sequence, maxlen = max_length, truncating =
trunc_type)
test_sentence = tokenizer.texts_to_sequences(test_sentence)
testing_padded = pad_sequences(test_sentence, maxlen = max_length)


# Obtain a screenshot/example of a padded sequence

print("padded sequence:", training_sentence[5])


padded sequence: I have to the plug to get it to line up right to get dece
nt volume


# We can group by function to observe how many different categories of
sentiment that we have. In this case, we have two:
# 0 = Negative, 1 = Positive
df.groupby('outlook').count()
```

B4. Number of Sentiment Categories

The number of sentiment categories in our current analysis are two, represented by 0 and 1, where 0 is equal to negative sentiment and 1 is equal to positive sentiment.

As for our activation function for the final dense layer of network through Keras will be by softmax. This is due to the fact that we would like the results to be interpretated as a probability distribution and return with an array of probability scores.

B5. Data Preparation Steps

The data preparation steps can be broken down as follows:

I. Loaded three different files being: Amazon, IMdb and Yelp into our dataframe. This was the case as to improve the training and data classification with these additional labelled data
II. Went and used labels as a primary relationship to combine the different data sets into one complete set that we use in our analysis
III. Encoded our different sentiments or ‘outlook’ in this case with either the binary values of 0 or 1. 0 represents the negative outlook and 1 represents the positive outlook.
IV. Summarized the data using info and describe statistical functions as well as addressed any null or missing values

V. Removed unnecessary characters to clean up the data and only leave in significant data for our analysis
VI. Removed both stop and infrequent wordings to narrow down the data. This can be also related to basic data cleaning
VII. Performed our training and testing sets by dividing our data through train and test functions
VIII. Created tokens for each of our reviews and derived meaning through derived program logic.
IX. Extract prepared dataset

B6. Extracted Data Set

```
#Extract our Data Set
df.to_csv('nlp_data_prep.csv')
```

Part III: Network Architecture

C1. Output of Model Summary

```
# We want to now use tensorflow to create our deep learning model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length =
max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'softmax')
])


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


print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 16)           32000
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 6)                 102
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 32,109
Trainable params: 32,109
Non-trainable params: 0
_________________________________________________________________
None
```

C2. Number of layers, Type of layers, and Total number of Parameters

In this model, we are making use of 4 layers with each type of layer explained below:

I. As shown in section C1, the first layer we have consists of the embedding layer in which we see that the total parameters present in this layer is 32 thousand. As for the output and shape, we can see that it has limits of 100 max sentence length and dimension of 16.
II. As shown in section C1, the second layer we have consists of the global average pooling 1D layer in which we see that the total parameters present in this layer is 0. The reason for this is because vectors are flattened from the first layer and translated to a one-dimensional plane.
III. As shown in section C1, the third layer we have consists of the Dense layer in which we see that the total parameters present in this layer is 102. This layer also contains 6 neurons.
IV. As shown in section C1, the fourth layer we have consists of the Dense layer as well, however this will be the layer that will be applying the softmax function. We see that this final layer has the total parameters present at 7.

Finally, out of the total parameters of 32,109 we see that 100% of them are trainable and that there are 0 that remain non-trainable.

C3. Hyperparameters Justification

I. Activation Function
   - Activation Function is a parameter that is present in each layer. This requires input data to be fed into our input layer which then goes through our hidden layer and finally the final output layer. It is important to note that the input data that goes through each of the layers changes according to the activation function **(Rendyk, 2021)**. In our case, we see that with an input of negative outlook data, we get an output represented of negative with the value of 0, otherwise 1.

II. Number of Nodes per Layer

- We have to use experimentation to obtain the best number of nodes to use for our particular data. This is another one of the hyperparameters that we must specify while in the process of configuring our network. In this case, considering the relatively small size of our data, we have 6 nodes among our third layer and 1 node in our final layer. This will be enough to obtain the results that we are looking for.

III. Loss Function

- For our loss function, we make use of the binary_crossentropy. The reason for this is because it is able to compute cross entropy loss between our true and predicted labels **(Keras, 2022)**. This is used for binary classification of either 0 or 1. This is exactly our case as our outputs will be in the form of binary results. It also shows probabilistic loss which we would like to have in our results.

IV. Optimizer

- We are using the Adam optimizer for this analysis. It is an algorithm that can be used in replacement of a classical stochastic gradient descent. The advantages of using the Adam is that it requires little memory, well suited for large problems in terms of data and parameters, require little tuning, and appropriate for problems containing lots of 'noise'.

V. Stopping Criteria

- An Early stopping criteria was implemented as to achieve the goal of minimizing loss. This criteria monitors the training and prevents over or under fitting by effectively stopping the training the moment a monitored metric has ceased to improve.

VI. Evaluation Metric

- The evaluation metric we used was simply to compare how accurate the sequential model was in terms of the proximity between our measured values and the values that are known.

Part IV: Model Evaluation

D1. Impact of Implementing Stopping Criteria

```
# Stopping Criteria
training_labels_final = np.array(training_label)
test_labels_final = np.array(test_label)


# Set 30 epoch
num_epochs = 30

# Fit model
history = model.fit(padded,
                    training_labels_final,
                    epochs = num_epochs,
                    validation_data = (testing_padded, test_labels_final))
```

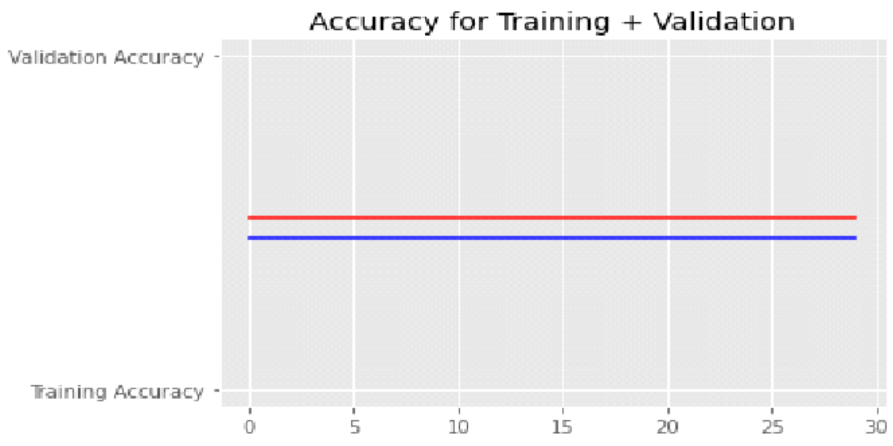
```
Epoch 1/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0219 - accu
racy: 0.5168 - val_loss: 1.6193 - val_accuracy: 0.4545
Epoch 2/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0218 - accu
racy: 0.5168 - val_loss: 1.7069 - val_accuracy: 0.4545
Epoch 3/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0217 - accu
racy: 0.5168 - val_loss: 1.6640 - val_accuracy: 0.4545
Epoch 4/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0209 - accu
racy: 0.5168 - val_loss: 1.8482 - val_accuracy: 0.4545
Epoch 5/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0238 - accu
racy: 0.5168 - val_loss: 1.6247 - val_accuracy: 0.4545
Epoch 6/30
69/69 [==============================] - 0s 4ms/step - loss: 0.0210 - accu
racy: 0.5168 - val loss: 1.6508 - val accuracy: 0.4545
Epoch 7/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0238 - accu
racy: 0.5168 - val_loss: 1.6115 - val_accuracy: 0.4545
Epoch 8/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0221 - accu
racy: 0.5168 - val_loss: 1.6233 - val_accuracy: 0.4545
Epoch 9/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0212 - accu
racy: 0.5168 - val_loss: 1.7869 - val_accuracy: 0.4545
Epoch 10/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0202 - accu
racy: 0.5168 - val_loss: 1.7165 - val_accuracy: 0.4545
Epoch 11/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0197 - accu
racy: 0.5168 - val_loss: 1.9404 - val_accuracy: 0.4545
Epoch 12/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0222 - accu
racy: 0.5168 - val_loss: 1.9057 - val_accuracy: 0.4545
Epoch 13/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0228 - accu
racy: 0.5168 - val_loss: 1.7292 - val_accuracy: 0.4545
```

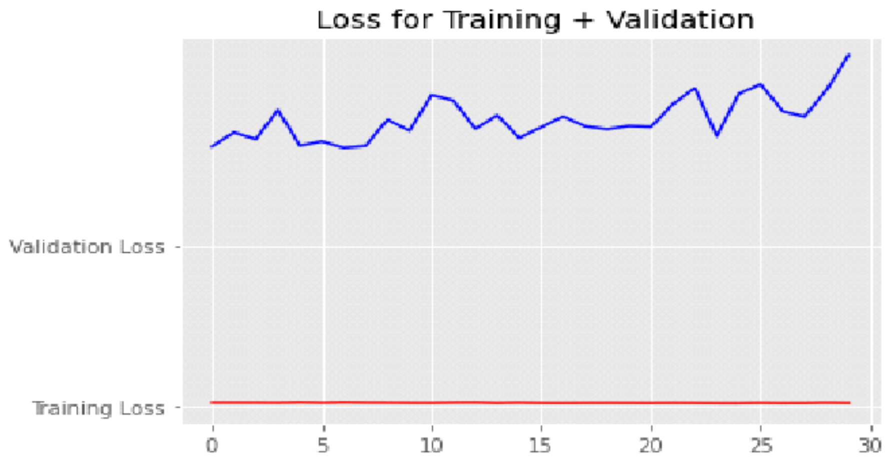
```
Epoch 15/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0216 - accu
racy: 0.5168 - val_loss: 1.6725 - val_accuracy: 0.4545
Epoch 16/30
69/69 [==============================] - 1s 8ms/step - loss: 0.0191 - accu
racy: 0.5168 - val_loss: 1.7387 - val_accuracy: 0.4545
Epoch 17/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0187 - accu
racy: 0.5168 - val_loss: 1.8077 - val_accuracy: 0.4545
Epoch 18/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0193 - accu
racy: 0.5168 - val_loss: 1.7445 - val_accuracy: 0.4545
Epoch 19/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0198 - accu
racy: 0.5168 - val_loss: 1.7266 - val_accuracy: 0.4545
Epoch 20/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0193 - accu
racy: 0.5168 - val_loss: 1.7466 - val_accuracy: 0.4545
Epoch 21/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0188 - accu
racy: 0.5168 - val_loss: 1.7411 - val_accuracy: 0.4545
Epoch 22/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0197 - accu
racy: 0.5168 - val_loss: 1.8832 - val_accuracy: 0.4545
Epoch 23/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0190 - accu
racy: 0.5168 - val_loss: 1.9837 - val_accuracy: 0.4545
Epoch 24/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0180 - accu
racy: 0.5168 - val_loss: 1.6836 - val_accuracy: 0.4545
Epoch 25/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0177 - accu
racy: 0.5168 - val_loss: 1.9471 - val_accuracy: 0.4545
Epoch 26/30
69/69 [==============================] - 0s 4ms/step - loss: 0.0200 - accu
racy: 0.5168 - val_loss: 2.0096 - val_accuracy: 0.4545
Epoch 27/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0181 - accu
racy: 0.5168 - val_loss: 1.8400 - val_accuracy: 0.4545
Epoch 28/30
69/69 [==============================] - 0s 6ms/step - loss: 0.0187 - accu
racy: 0.5168 - val_loss: 1.8058 - val_accuracy: 0.4545
Epoch 29/30
69/69 [==============================] - 0s 4ms/step - loss: 0.0208 - accu
racy: 0.5168 - val_loss: 1.9713 - val_accuracy: 0.4545
Epoch 30/30
69/69 [==============================] - 0s 5ms/step - loss: 0.0194 - accu
racy: 0.5168 - val_loss: 2.1871 - val_accuracy: 0.4545
```

As we can see above, it is beneficial to implement the stopping criteria because the accuracy after the first epoch throughout the 30th epoch remains the same at 0.5168. While this is happening, we can also see that our loss is gradually decreasing from 0.0219 all the way to 0.0194 after are 30th epoch. Therefore, with accuracy staying the same while loss diminishes, we do not need to use all 30 of the epoch. This diminishment can be related to the model trying to over or underfit the data. This is why it is beneficial to implement a stopping criteria.

D2. Visualization of Model Training Process

```
# Create Visualization for our Training Process
plt.style.use('ggplot')

accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs, accuracy, 'r', 'Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', 'Validation Accuracy')
plt.title('Accuracy for Training + Validation')
plt.figure()
plt.plot(epochs, loss, 'r', 'Training Loss')
plt.plot(epochs, val_loss, 'b', 'Validation Loss')
plt.title(' Loss for Training + Validation')
plt.figure()
```





```
# Function to evaluate the accuracy of our model
loss, accuracy = model.evaluate(padded, training_labels_final, verbose = 0)
print('Accuracy: %f' % (accuracy))
```

```
Accuracy: 0.516833
```

D3. Fitness of Model

One of the main issues that are present with training neural networks is regarding the number of training epochs to utilize. As briefly mentioned in section D1, too many epochs will inevitably lead to overfitting of the data, and likewise too little epochs will lead to underfitting. Due to our early stopping criteria, we stopped at the first epoch as the accuracy remained the same after that, but loss slowly diminished in value. Thus, using more than the first epoch to train would have led to overfitting the data.

We can also consider overfitting due to our large vocab size (2,000). The more data that is injected into our model, the more complex the model could become and therefore could also run the risk of causing overfitting to occur. Thus another way of tackling the problem of having our data overfitted is to possible reduce our vocabulary size.

D4. Predictive Accuracy of Trained Network

With the results we got in section D1, we observe that the resulting accuracy for our training and testing data sets consisted of an accuracy of 51.68% along with a validation accuracy of 45.45%. This is taking into account our stopping criteria triggering on the first epoch. This means that unfortunately the probability that our classification on our predicted customer outlook being either positive of negative is relatively low. Ideally we would like the accuracy range to be between 80%-90%.

Part V: Summary and Recommendations

E. Predictive Accuracy of Trained Network

Codes in Jupyter Notebooks is attached to this assignment for your consideration

F. Functionality of Neural Network

The functionality of this neural network unfortunately cannot be recommended to be used based on its low scores for both accuracy and validation accuracy. The purpose of this neural network was to be able to build a model that would accurately be able to predict either positive or negative customer sentiment by undergoing training on the 3 combined labelled datasets.

We would have like this neural network to have the ability to use these training sets and with confidence, predict customer outlook on unlabelled data that we collect. Even with incorporating efficient hyperparameters in our four layers along with implementing a stopping criterion, our NLP model falls short of being able to predict with confidence. Having accuracy hovering around the 50% for both accuracy and validation accuracy is quite far away from where we would be able to deem the functionality of the network suitable at a level of around 80%-90%.

G. Recommendations

Recommendation would be to not use this model as it has an accuracy percentage of only 51.68%. This would not ensure confidence in our prediction results when applied to unlabelled data, as so we should implement some retraining with the data until we are able to reach a suitable accuracy rate of around 80%-90% at least. As our model consisted of labelled data that crosses various industries(Amazon = ecommerce, IMDB = movies, Yelp = Food), perhaps accuracy would increase if we focused on industries that have more relation to each other. That way the specific vocabulary would be used as a consensus in more of the same way or similar meaning. An example of this is if we were to get reviews from Amazon, Ebay, and Aliexpress.

Logically this makes sense and if trained with our NLP model, perhaps we would be able to more accuracy predict customer outlooks on products for that specific industry.

Part VI: Reporting

H. Jupyter Notebook Reporting

A copy of the Jupyter Notebook used for this analysis is currently attached to the assignment for you consideration.

I. Third-Party Code for Supporting Application

Shubham Singh (2019). How To Get Started with NLP – 6 Unique Methods To Perform Tokenization. https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/

Jason Brownlee (2017). How To Clean Text for Machine Learning with Python. https://machinelearningmastery.com/clean-text-machine-learning-python/

TensorFlow (2022). Masking and Padding with Keras. https://www.tensorflow.org/guide/keras/masking_and_padding

Scikit-Learn (2022). Early Stopping of Stochastic Gradient Descent. https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_early_stopping.html

J. Acknowledged Sources

Tal Perry (2021). What is Tokenization in Natural Language Processing (NLP)? https://www.machinelearningplus.com/nlp/what-is-tokenization-in-natural-language-processing/

Data Flair (2022). Masking and Padding in Keras. https://data-flair.training/blogs/masking-and-padding-in-keras/#:~:text=To%20ensure%20that%20all%20the,data%20points%20of%20standardized%20tensors.&text=All%20the%20input%20to%20this,must%20have%20length%20equal%20lenght_sequence.

Rendyk (2021). Tuning the Hyperparameters and Layers of Neural Network Deep Learning. https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/#:~:text=An%20activation%20function%20is%20a,according%20to%20the%20activation%20function.

Keras (2022). Probabilistic Losses. https://keras.io/api/losses/probabilistic_losses/