D213 – ADVANCED DATA ANALYTICS

Task 2: Sentiment Analysis using Neural Networks

WGU - MSDA

Advanced Data Analytics with Churn Dataset

Richard Flores
rflo147@wgu.edu
Student ID: 006771163

Table of Contents

Part I: Research Question

A1. Research Question

The purpose of this Research assignment is to create a neural network to analyze word usage and context using Natural Language Processing techniques. The end result will include visualizations and a report in an interactive development environment.

In this analysis we will answer the question:

Is it possible to gain insightful sentiment metrics from the UCI Sentiment Labeled Sentences dataset containing customer reviews by using a Natural Language Processing Neural Network?

A2. Objectives and Goals

The objective of this assessment is to create a NLP Neural Network to analyze the UCI Sentiment Labeled Sentences dataset. The dataset contains three subsets of customer reviews from wildly popular review sites including IMDB, Amazon, and Yelp. We can accomplish this objective by utilizing the TensorFlow Python library to analyze and forecast negative or positive sentiment. The overall goal of the analysis is to assist stakeholders and executives in making informed decisions based on the analyzed metrics of customer preference by using text classification to produce useful predictions.

A3. Prescribed Network

To complete our stated goal of performing text classification and text sequence prediction in this analysis we will be using the Python TensorFlow library and in addition Keras for Natural Language Processing or NLP.

TensorFlow is an open-source Python Machine Learning library which uses the higher-level API Keras to conduct text classification. In our Neural Network we will import TensorFlow and Keras as well as the Tokenizer API to create word encodings and assign integer value to words passed in the text (Kausar 2019). We can use the manipulated list sequences to train the Neural Network and derive our metrics which can be translated into useful insights.

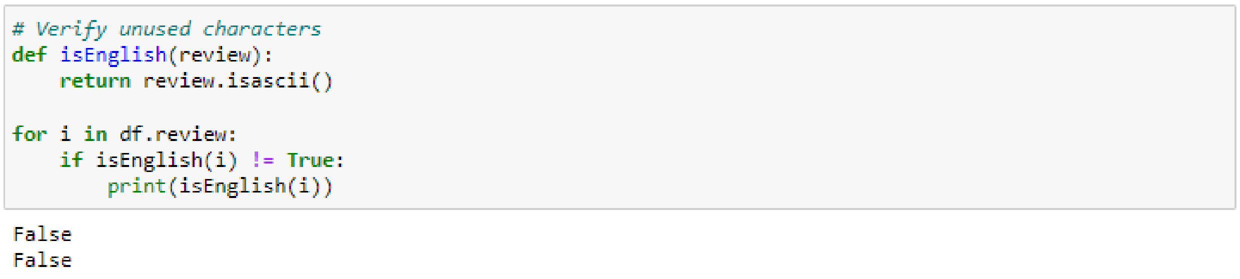
Part II: Data Preparation

B1. Data Exploration

<u>Summary of Data Cleaning Process</u>

1. Presence of unusual characters (e.g., emojis, non-English characters, etc.)

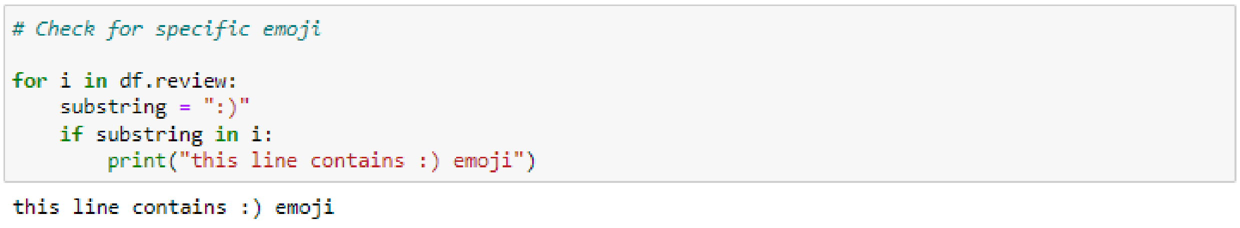
   We have verified the non-existence of non-English characters and miscellaneous ascii characters:



```
# Verify unused characters
def isEnglish(review):
    return review.isascii()

for i in df.review:
    if isEnglish(i) != True:
        print(isEnglish(i))
```

```
False
False
```

   But we did encounter the presence of the following emoji and have taken steps to clean the data:

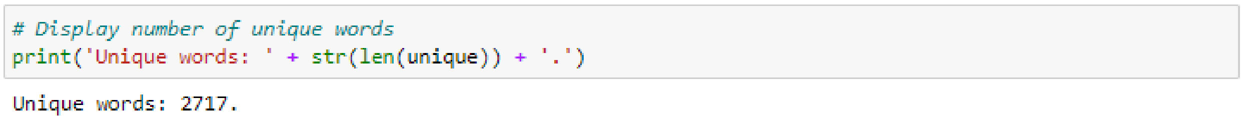


```
# Check for specific emoji

for i in df.review:
    substring = ":)"
    if substring in i:
        print("this line contains :) emoji")
```

```
this line contains :) emoji
```

2. Vocabulary size

   The calculated vocabulary size was determined as follows:

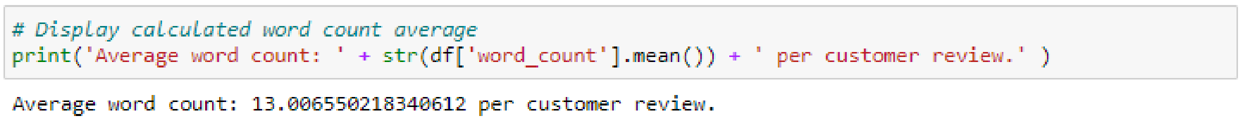


```
# Display number of unique words
print('Unique words: ' + str(len(unique)) + '.')
```

```
Unique words: 2717.
```

3. Proposed word embedding length

   From our exploratory analysis we have determined the word embedding length as follows:



```
# Display calculated word count average
print('Average word count: ' + str(df['word_count'].mean()) + ' per customer review.' )
```

```
Average word count: 13.006550218340612 per customer review.
```

4. Statistical justification for the chosen maximum sequence length

The maximum sequence length is based on the following metrics:

```
# Display calculated shortest length
min(length_reviews)
```

```
7
```

```
# Display calculated sum of review length
sum(length_reviews)
```

```
196560
```

```
# Display longest review length
print('Max Review Length: ' + str(max(length_reviews)) + ' words.')
```

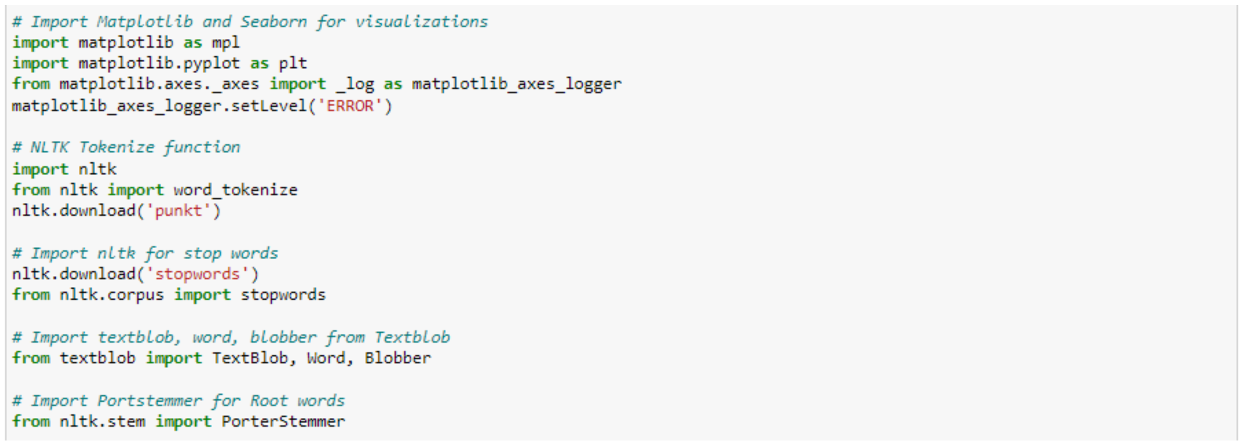
```
Max Review Length: 7944 words.
```

```
# Display created vector array
X_df.head(10)
```

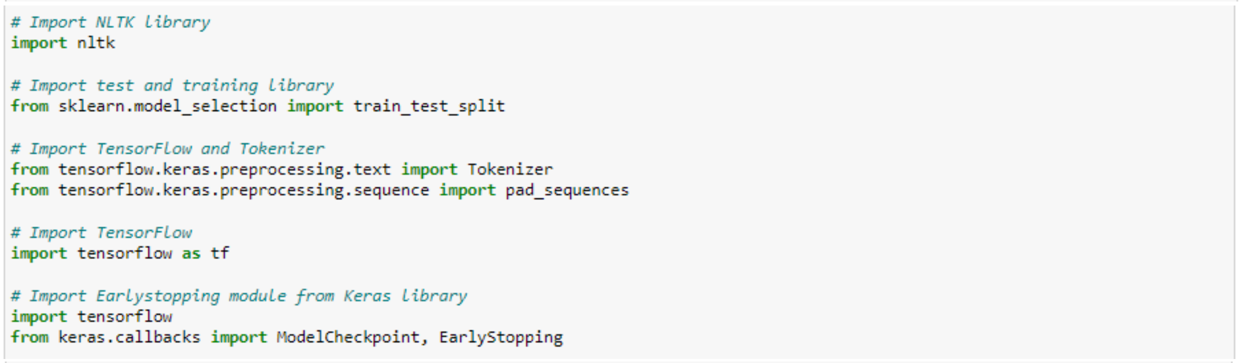
	and	but	for	good	great	in	is	it	my	not	of	on	that	the	this	to	very	was	with	you
0	0	0	1	0	0	2	1	1	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	4	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

CODE

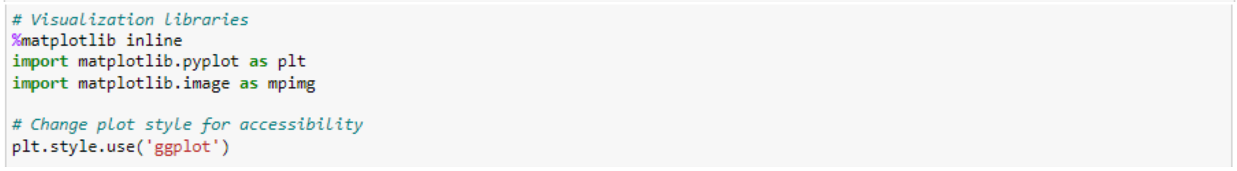
```
# Import necessary libraries for Data Analysis and creating the Neural Network
# Data Analysis Libraries
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

# Import TensorFlow for NLP
from tensorflow import keras

# Import Dateutil to edit parse tree and expressions
from dateutil.parser import parse

# Import functools reduce library
from functools import reduce

# Import CountVectorizer for tokenization
from sklearn.feature_extraction.text import CountVectorizer

# Import RE for text cleanup
import re
```



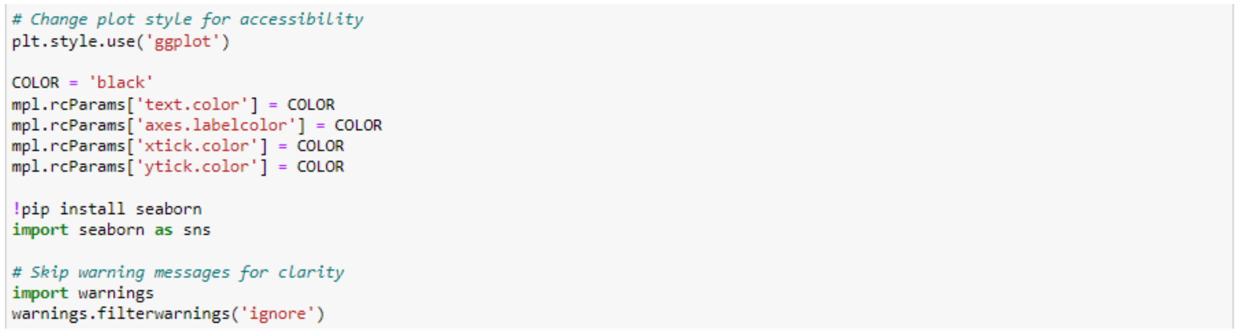
```
# Import Matplotlib and Seaborn for visualizations
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')

# NLTK Tokenize function
import nltk
from nltk import word_tokenize
nltk.download('punkt')

# Import nltk for stop words
nltk.download('stopwords')
from nltk.corpus import stopwords

# Import textblob, word, blobber from Textblob
from textblob import TextBlob, Word, Blobber

# Import Portstemmer for Root words
from nltk.stem import PorterStemmer
```



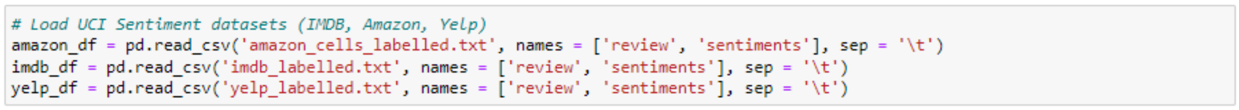
```
# Import NLTK library
import nltk

# Import test and training library
from sklearn.model_selection import train_test_split

# Import TensorFlow and Tokenizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Import TensorFlow
import tensorflow as tf

# Import Earlystopping module from Keras library
import tensorflow
from keras.callbacks import ModelCheckpoint, EarlyStopping
```



```
# Visualization libraries
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Change plot style for accessibility
plt.style.use('ggplot')
```



```
# Change plot style for accessibility
plt.style.use('ggplot')

COLOR = 'black'
mpl.rcParams['text.color'] = COLOR
mpl.rcParams['axes.labelcolor'] = COLOR
mpl.rcParams['xtick.color'] = COLOR
mpl.rcParams['ytick.color'] = COLOR

!pip install seaborn
import seaborn as sns

# Skip warning messages for clarity
import warnings
warnings.filterwarnings('ignore')
```



```
# Load UCI Sentiment datasets (IMDB, Amazon, Yelp)
amazon_df = pd.read_csv('amazon_cells_labelled.txt', names = ['review', 'sentiments'], sep = '\t')
imdb_df = pd.read_csv('imdb_labelled.txt', names = ['review', 'sentiments'], sep = '\t')
yelp_df = pd.read_csv('yelp_labelled.txt', names = ['review', 'sentiments'], sep = '\t')
```

```python
# Verify dataframes by printing headers
print(amazon_df.head(10))
print(imdb_df.head(10))
print(yelp_df.head(10))
```

```
                                              review  sentiments
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
5  I have to jiggle the plug to get it to line up...           0
6  If you have several dozen or several hundred c...           0
7        If you are Razr owner...you must have this!           1
8                Needless to say, I wasted my money.           0
9                   What a waste of money and time!.           0
                                              review  sentiments
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
5  The rest of the movie lacks art, charm, meanin...           0
6                                Wasted two hours.             0
7  Saw the movie today and thought it was a good ...           1
8                              A bit predictable.              0
9  Loved the casting of Jimmy Buffet as the scien...           1
                                              review  sentiments
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
5     Now I am getting angry and I want my damn pho.           0
6              Honeslty it didn't taste THAT fresh.)           0
7  The potatoes were like rubber and you could te...           0
8                          The fries were great too.           1
9                                     A great touch.           1
```

```python
# Confirm records in Amazon dataset
amazon_df.info
```

```
<bound method DataFrame.info of                                                   review  sentiments
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
..                                                 ...         ...
995  The screen does get smudged easily because it ...           0
996  What a piece of junk.. I lose more calls on th...           0
997                      Item Does Not Match Picture.           0
998  The only thing that disappoint me is the infra...           0
999  You can not answer calls with the unit, never ...           0

[1000 rows x 2 columns]>
```

```python
# Confirm records in IMDB dataset
imdb_df.info
```

```
<bound method DataFrame.info of                                                   review  sentiments
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
..                                                 ...         ...
743  I just got bored watching Jessice Lange take h...           0
744  Unfortunately, any virtue in this film's produ...           0
745                    In a word, it is embarrassing.              0
746                                Exceptionally bad!              0
747  All in all its an insult to one's intelligence...           0

[748 rows x 2 columns]>
```

```
# Confirm records in Yelp dataset
yelp_df.info
```

```
<bound method DataFrame.info of                                               review  sentiments
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
..                                                 ...         ...
995  I think food should have flavor and texture an...           0
996                           Appetite instantly gone.           0
997  Overall I was not impressed and would not go b...           0
998  The whole experience was underwhelming, and I ...           0
999  Then, as if I hadn't wasted enough of my life ...           0

[1000 rows x 2 columns]>
```

```
# Create dataframe from supplied datasets
data_frames = [amazon_df, imdb_df, yelp_df]
```

```
# Reduce datasets into single dataframe
df = reduce(lambda  left,right: pd.merge(left,right, on = ['review', 'sentiments'], how = 'outer'), data_frames)
```

```
# Verify missing records in dataset
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
review        0
sentiments    0
dtype: int64
```

```
# Verify binary sentiment values take only 0 or 1 values
df.sentiments.unique()
```

```
array([0, 1], dtype=int64)
```

```
# Count total number of Positive and Negative sentiments present
print('Dataframe Sentiments \n', df.sentiments.value_counts())
```

```
Dataframe Sentiments 
 1    1386
0    1362
Name: sentiments, dtype: int64
```

```
# Verify missing records in dataset
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
review        0
sentiments    0
dtype: int64
```

```
# Verify binary sentiment values take only 0 or 1 values
df.sentiments.unique()
```

```
array([0, 1], dtype=int64)
```

```
# Count total number of Positive and Negative sentiments present
print('Dataframe Sentiments \n', df.sentiments.value_counts())
```

```
Dataframe Sentiments 
 1    1386
0    1362
Name: sentiments, dtype: int64
```

```
# Total Percentages of Positive and Negative sentiments
print('Merged Dataframe \n', df.sentiments.value_counts() / len(df))
```

```
Merged Dataframe 
 1    0.504367
0    0.495633
Name: sentiments, dtype: float64
```

```
# Create list of containing length of review characters
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
```

```
# Display calculated shortest length
min(length_reviews)
```

```
7
```

```
# Display calculated sum of review length
sum(length_reviews)
```

```
196560
```

```
# Display longest review length
print('Max Review Length: ' + str(max(length_reviews)) + ' words.')
```

```
Max Review Length: 7944 words.
```

```
# Create tokenization
# Set max features to 20 words
vect = CountVectorizer(max_features=20)

# Set vectorizer Fit
vect.fit(df.review)
```

```
CountVectorizer(max_features=20)
```

```
# Create vector transform
X = vect.transform(df.review)
X
```

```
<2748x20 sparse matrix of type '<class 'numpy.int64'>'
	with 8115 stored elements in Compressed Sparse Row format>
```

```
# Create array from transformed vector
my_array = X.toarray()
```

```
# Tranfer data into Pandas Dataframe
X_df = pd.DataFrame(my_array, columns = vect.get_feature_names())
```

```
# Display created vector array
X_df.head(10)
```

	and	but	for	good	great	in	is	it	my	not	of	on	that	the	this	to	very	was	with	you
0	0	0	1	0	0	2	1	1	0	0	0	0	0	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	4	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

```
# Create ngram range
min_n = 1
max_n = 3
vect = CountVectorizer(ngram_range=(min_n, max_n))
```

```python
# Iterate reviews using list comprehension
word_tokens = [word_tokenize(review) for review in df.review]
```

```python
# Print initial 5 token reviews
print(word_tokens[0:5])
```

```
[['So', 'there', 'is', 'no', 'way', 'for', 'me', 'to', 'plug', 'it', 'in', 'here', 'in', 'the', 'US', 'unless', 'I', 'go', 'b
y', 'a', 'converter', '.'], ['Good', 'case', ',', 'Excellent', 'value', '.'], ['Great', 'for', 'the', 'jawbone', '.'], ['Tied',
'to', 'charger', 'for', 'conversations', 'lasting', 'more', 'than', '45', 'minutes.MAJOR', 'PROBLEMS', '!', '!'], ['The', 'mi
c', 'is', 'great', '.']]
```

```python
# Create token length list and iterate over lengths
len_tokens = []

for i in range(len(word_tokens)):
    len_tokens.append(len(word_tokens[i]))
```

```python
# Verify unused characters
def isEnglish(review):
    return review.isascii()

for i in df.review:
    if isEnglish(i) != True:
        print(isEnglish(i))
```

```
False
```

```python
# Check for specific emoji

for i in df.review:
    substring = ":)"
    if substring in i:
        print("this line contains :) emoji")
```

```
this line contains :) emoji
```

```python
# Create function that counts words in each review and insert into word_count dataframe
def count_words(review):
    words = review.split()
    return len(words)

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

```python
# Display calculated word count average
print('Average word count: ' + str(df['word_count'].mean()) + ' per customer review.' )
```

```
Average word count: 13.006550218340612 per customer review.
```

```python
# Calculate vocabulary size and unique entries
unique = set(df['review'].str.replace('[^a-zA-Z]', '').str.lower().str.split(' ').sum())
```

```python
# Display number of unique words
print('Unique words: ' + str(len(unique)) + '.')
```

```
Unique words: 2717.
```

```python
# Use RE to clean hyperlinks and create cleaned dataframe
def clean_url(review_text):
    return re.sub('r.http\S+', '', review_text)

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

```python
# Clean punctuation from text and update cleaned dataframe
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

df['clean_review'] = clean(df['clean_review'])
```

```python
# Clean irrelevant characters and update cleaned dataframe
def clean_non_alphanumeric(review_text):
    return re.sub('[^a-zA-Z]', ' ', review_text)

df['clean_review'] = df['clean_review'].apply(clean_non_alphanumeric)
```

```python
# Transform all strings to lowercase and update dataframe
def clean_case(review_text):
    return str(review_text).lower()

df['clean_review'] = df['clean_review'].apply(clean_case)
df
```

	review	sentiments	word_count	clean_review
0	So there is no way for me to plug it in here i...	0	21	so there is no way for me to plug it in here i...
1	Good case, Excellent value.	1	4	good case excellent value
2	Great for the jawbone.	1	4	great for the jawbone
3	Tied to charger for conversations lasting more...	0	11	tied to charger for conversations lasting more...
4	The mic is great.	1	4	the mic is great
...	...	...	...	...
2743	I think food should have flavor and texture an...	0	12	i think food should have flavor and texture an...
2744	Appetite instantly gone.	0	3	appetite instantly gone
2745	Overall I was not impressed and would not go b...	0	10	overall i was not impressed and would not go b...
2746	The whole experience was underwhelming, and I ...	0	16	the whole experience was underwhelming and i ...
2747	Then, as if I hadn't wasted enough of my life ...	0	28	then as if i hadn t wasted enough of my life ...

2748 rows × 4 columns

```python
# Set stop words
stop = stopwords.words('english')
df['clean_review'] = df['clean_review'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
df
```

	review	sentiments	word_count	clean_review
0	So there is no way for me to plug it in here i...	0	21	way plug us unless go converter
1	Good case, Excellent value.	1	4	good case excellent value
2	Great for the jawbone.	1	4	great jawbone
3	Tied to charger for conversations lasting more...	0	11	tied charger conversations lasting minutes maj...
4	The mic is great.	1	4	mic great
...	...	...	...	...
2743	I think food should have flavor and texture an...	0	12	think food flavor texture lacking
2744	Appetite instantly gone.	0	3	appetite instantly gone
2745	Overall I was not impressed and would not go b...	0	10	overall impressed would go back
2746	The whole experience was underwhelming, and I ...	0	16	whole experience underwhelming think go ninja ...
2747	Then, as if I hadn't wasted enough of my life ...	0	28	wasted enough life poured salt wound drawing t...

2748 rows × 4 columns

```python
# Clear less frequent words and update dataset
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))
```

```
# Use textblob to clean spelling errors
df['clean_review'].apply(lambda x: str(TextBlob(x).correct()))
df
```

	review	sentiments	word_count	clean_review
0	So there is no way for me to plug it in here i...	0	21	way plug us unless go
1	Good case, Excellent value.	1	4	good case excellent value
2	Great for the jawbone.	1	4	great jawbone
3	Tied to charger for conversations lasting more...	0	11	charger conversations lasting minutes major pr...
4	The mic is great.	1	4	mic great
...	...	...	...	...
2743	I think food should have flavor and texture an...	0	12	think food flavor texture lacking
2744	Appetite instantly gone.	0	3	gone
2745	Overall I was not impressed and would not go b...	0	10	overall impressed would go back
2746	The whole experience was underwhelming, and I ...	0	16	whole experience underwhelming think go sushi ...
2747	Then, as if I hadn't wasted enough of my life ...	0	28	wasted enough life salt time took bring check

2748 rows × 4 columns

```
# Stem words and update dataframe
st = PorterStemmer()

df['clean_review'] = df['clean_review'].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
```

```
# Initiate lemmatization and update dataframe
nltk.download('wordnet')

df['clean_review'] = df['clean_review'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Richard\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# Remove punctuation and update dataframe
df['clean_review'] = df['clean_review'].str.replace('[^\w\s]', '')
```

```
# Update polarity
df['polarity'] = df['clean_review'].map(lambda text: TextBlob(text).sentiment.polarity)
df
```

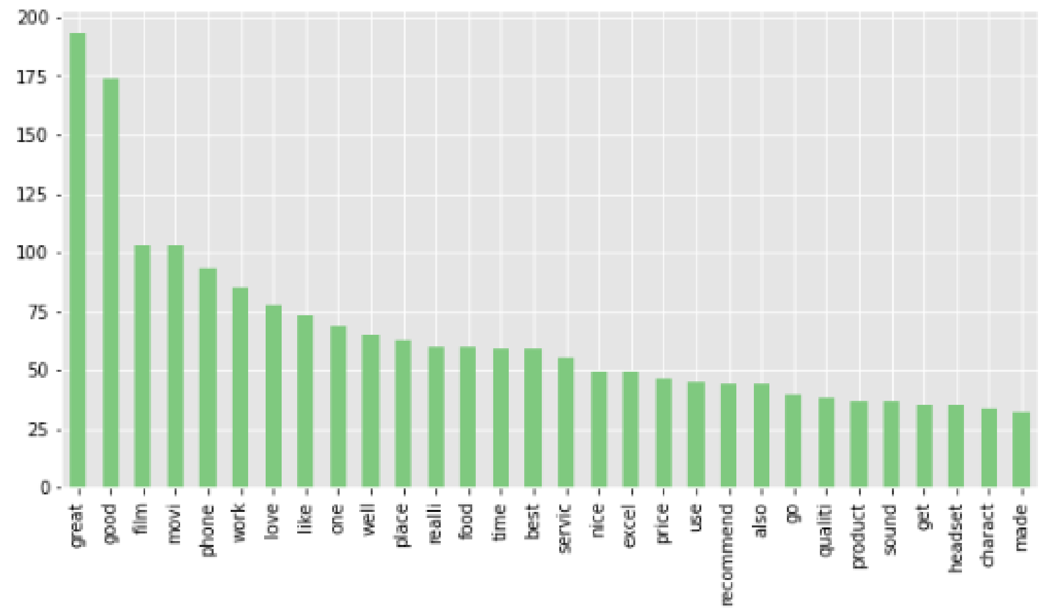
	review	sentiments	word_count	clean_review	polarity
0	So there is no way for me to plug it in here i...	0	21	way plug u unless go	0.00000
1	Good case, Excellent value.	1	4	good case excel valu	0.70000
2	Great for the jawbone.	1	4	great jawbon	0.80000
3	Tied to charger for conversations lasting more...	0	11	charger convers last minut major problem	0.03125
4	The mic is great.	1	4	mic great	0.80000
...	...	...	...	...	...
2743	I think food should have flavor and texture an...	0	12	think food flavor textur lack	0.00000
2744	Appetite instantly gone.	0	3	gone	0.00000
2745	Overall I was not impressed and would not go b...	0	10	overal impress would go back	0.00000
2746	The whole experience was underwhelming, and I ...	0	16	whole experi underwhelm think go sushi next time	0.10000
2747	Then, as if I hadn't wasted enough of my life ...	0	28	wast enough life salt time took bring check	0.00000

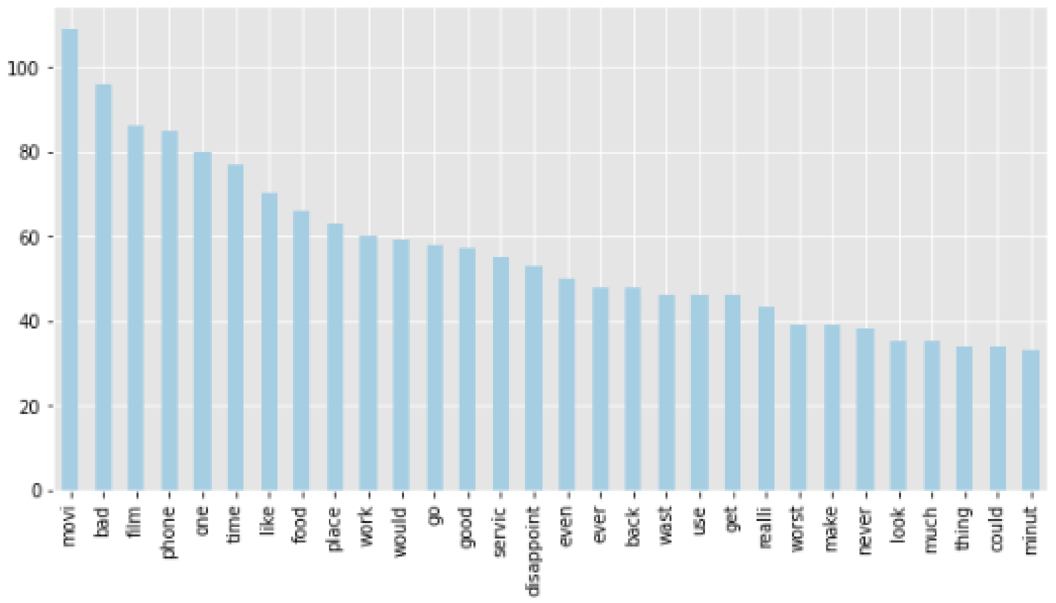
2748 rows × 5 columns

```
# Set negative and positive sentiment visualizations
negative_review = df[df.sentiments == 0]['clean_review']
positive_review = df[df.sentiments == 1]['clean_review']
```

```
# Create word frequency lists for bar graph
color = ['Accent', 'Paired']
split_df = [positive_review, negative_review]
```

```
# Display bar graphs for text lemmas
for item in range(2):
    plt.figure(figsize = (10, 5))
    pd.Series(' '.join([i for i in split_df[item]]).split()).value_counts().head(30).plot(kind = 'bar', colormap = color[item])
    plt.show();
```





```
# Specify neutral words
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. Tokenization

Summary of Tokenization

Tokenization is an essential process of the Natural Language Processing Neural Network model. Tokenization is the process by which AI understands the meaning of the text inputted into the model. The goal of Tokenization is to prepare the text for later analysis in Count Vectorization and Deep Learning. To accomplish this goal the tokenization algorithm processes text into smaller units which we reference as tokens. The tokens fall into categories of words, characters, and subwords or n-gram characters (Vidhya 2021).

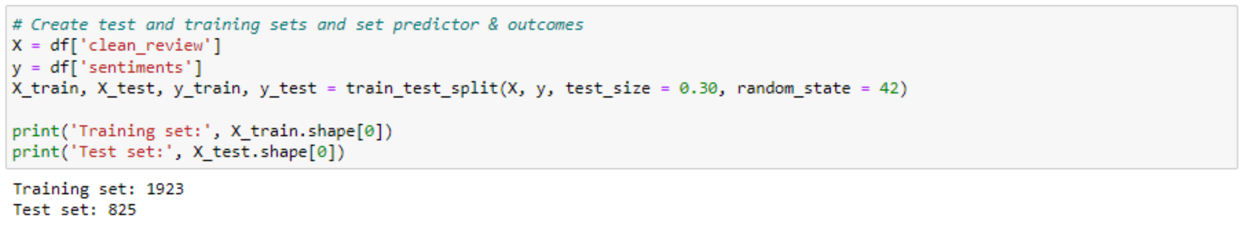
The steps taken for Tokenization of the text data are:

- Create test and training sets via data split into 80/20 portions
- Create Predictor and Outcome variables
- Create a tokenizer object
- Create four lists for analysis including training and test sentences, and training and test labels
- Create vocabulary parameters and fit text into tokenizer

The packages used for this process include:

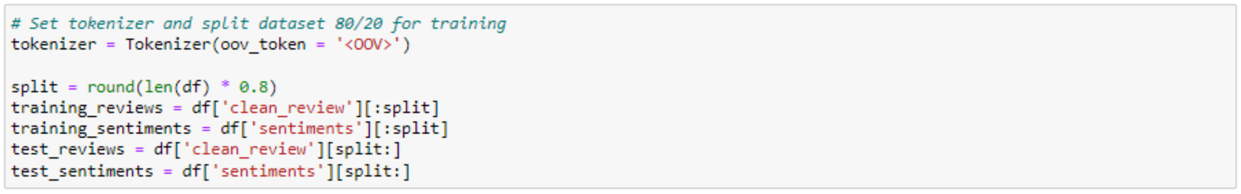
- From Scikit-learn the train_test_split library
- From TensorFlow Keras the Tokenizer and pad_sequences libraries

CODE



```
# Create test and training sets and set predictor & outcomes
X = df['clean_review']
y = df['sentiments']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 42)

print('Training set:', X_train.shape[0])
print('Test set:', X_test.shape[0])
```

```
Training set: 1923
Test set: 825
```



```
# Set tokenizer and split dataset 80/20 for training
tokenizer = Tokenizer(oov_token = '<OOV>')

split = round(len(df) * 0.8)
training_reviews = df['clean_review'][:split]
training_sentiments = df['sentiments'][:split]
test_reviews = df['clean_review'][split:]
test_sentiments = df['sentiments'][split:]
```

```
# Create training and test lists and append values
training_sentences = []
training_labels = []
test_sentences = []
test_labels = []

for row in training_reviews:
    training_sentences.append(str(row))
for row in training_sentiments:
    training_labels.append(row)
for row in test_reviews:
    test_sentences.append(str(row))
for row in test_sentiments:
    test_labels.append(row)
```

```
# Create vocabulary parameters and fit training set to tokenizer
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_sentences)
word_index = tokenizer.word_index
```

```
# Display index
print("Words:\n", word_index)
```

```
Words:
 {'<OOV>': 1, 'movi': 2, 'film': 3, 'good': 4, 'phone': 5, 'great': 6, 'work': 7, 'one': 8, 'like': 9, 'time': 10, 'bad': 11,
'use': 12, 'well': 13, 'realli': 14, 'love': 15, 'would': 16, 'place': 17, 'make': 18, 'even': 19, 'servic': 20, 'go': 21, 'g
et': 22, 'best': 23, 'ever': 24, 'product': 25, 'qualiti': 26, 'look': 27, 'charact': 28, 'also': 29, 'sound': 30, 'food': 3
1, 'headset': 32, 'recommend': 33, 'made': 34, 'watch': 35, 'excel': 36, 'batteri': 37, 'act': 38, 'see': 39, 'could': 40, 'n
ever': 41, 'better': 42, 'wast': 43, 'price': 44, 'thing': 45, 'disappoint': 46, 'back': 47, 'ear': 48, 'think': 49, 'year':
50, 'first': 51, 'much': 52, 'case': 53, 'way': 54, 'come': 55, 'nice': 56, 'still': 57, 'end': 58, 'play': 59, 'worst': 60,
'scene': 61, 'tri': 62, 'stori': 63, 'problem': 64, 'right': 65, 'say': 66, 'wonder': 67, 'littl': 68, 'real': 69, 'pretti':
70, 'everyth': 71, 'everi': 72, 'peopl': 73, 'call': 74, 'got': 75, 'actor': 76, 'minut': 77, 'two': 78, 'enough': 79, 'fee
l': 80, 'know': 81, 'plot': 82, 'money': 83, 'piec': 84, 'terribl': 85, 'suck': 86, 'show': 87, 'comfort': 88, 'life': 89, 'w
ant': 90, 'buy': 91, 'lot': 92, 'give': 93, 'seen': 94, 'new': 95, 'fit': 96, 'take': 97, 'noth': 98, 'line': 99, 'day': 100,
'perform': 101, 'screen': 102, 'worth': 103, 'mani': 104, 'cast': 105, 'script': 106, 'far': 107, 'charg': 108, 'poor': 109,
'camera': 110, 'order': 111, 'definit': 112, 'charger': 113, 'easi': 114, 'quit': 115, 'anyon': 116, 'amaz': 117, 'total': 11
8, 'highli': 119, 'long': 120, 'part': 121, 'enjoy': 122, 'last': 123, 'bought': 124, 'car': 125, 'fine': 126, 'find': 127,
'purchas': 128, 'expect': 129, 'hour': 130, 'funni': 131, 'star': 132, 'interest': 133, 'happi': 134, 'item': 135, 'found': 1
36, 'thought': 137, 'howev': 138, 'turn': 139, 'bluetooth': 140, 'seem': 141, 'big': 142, 'music': 143, 'job': 144, 'anoth':
145, 'alway': 146, 'aw': 147, 'direct': 148, 'bore': 149, 'pictur': 150, 'drop': 151, 'cool': 152, 'beauti': 153, 'black': 15
4, 'recept': 155, 'around': 156, 'kind': 157, 'horribl': 158, 'came': 159, 'effect': 160, 'stupid': 161, 'write': 162, 'impre
ss': 163, 'start': 164, 'though': 165, 'absolut': 166, 'need': 167, 'complet': 168, 'cheap': 169, 'incred': 170, 'experi': 17
```

```
print('Total words in vocabulary: ' + str(len(word_index)) + ' words.')
```

```
Total words in vocabulary: 1647 words.
```

B3. Padding Process

Padding Summary

The padding process is essential for our Neural Network as it is a requirement inputs have equal shape and size. From our exploratory analysis we observed reviews with varying sizes and to rectify this issue we will use the padding technique. The padding process adds zeroes to the end of a sequence to ensure all samples have equal sizes (Caner 2020). In this analysis we will add padding to the end of a sequence rather than the beginning for a determined minimum length of 100.

CODE

```
# Create padding for sentences
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences, maxlen = max_length, truncating = trunc_type)
test_sentences = tokenizer.texts_to_sequences(test_sentences)
testing_padded = pad_sequences(test_sentences, maxlen = max_length)
```

```
# Display sentences and padding
print("\nTraining sequences:\n", training_sentences)
```

```
Training sequences:
 ['way plug u unless go', 'good case excel valu', 'great jawbon', 'charger convers last minut major problem', 'mic great', 'p
lug get line right get decent volum', 'sever dozen sever contact imagin fun send one one', 'razr owner must', 'needle say was
t money', 'wast money time', 'sound qualiti great', 'impress go origin batteri extend batteri', 'two start notic static sound
headset', 'good qualiti though', 'design odd ear clip comfort', 'highli recommend one blue tooth phone', 'advis everyon', 'fa
r good', 'work great', 'work great', 'place way make wonder long would last', 'went motorola websit could get pair', 'bought
use fire absolut love', '', 'yet run new batteri two bar three day without charg', 'bought mother problem batteri', 'great po
cket pc phone combin', 'own phone month say best mobil phone', 'think instruct provid help', 'peopl hear talk pull talk phon
e', 'hold charg', 'simpl littl phone use', 'product peopl like ear', 'unus move car speed', 'two year left contract hate phon
e', 'car charger well charger includ make sure never run juic recommend', 'need least get phone book time first turn phone ba
tteri life short', 'kept well', 'poor talk time perform', 'case great work fine', 'worthless product', 'great camera that mp
nice clear great pictur qualiti', 'impress product', 'nice headset price right', 'hear garbag audio', 'excel bluetooth headse
t', 'featur want', 'right mind gonna buy batteri', 'verizon regard drop call return phone two day', 'case seem well made', 'd
isappoint batteri', 'loud enough turn like', 'good protect make phone', 'keyboard actual turn pda real world machin instead n
eat gadget', 'phone pretti sturdi never larg problem', 'love thing', 'everyth fine reason price e', 'disappoint', 'even drop
phone second still work great', 'happi complaint one regard sound qualiti end', 'button bad', 'essenti forget microsoft tech
support', 'realli recommend sinc look nice eleg cool', 'headphon great find think perhap best purchas made last sever year se
rious', 'buy differ phone', 'hold phone particular angl parti hear clearli', 'one big mp player button phone front cover let
```

```
# Display length of padded test sentences
print("\nTraining sequences:\n", padded)
print("\nTraining shape:", padded.shape)
print("Training type:", type(test_sentences))
print("Padded Training type:", type(padded))
```

```
Training sequences:
 [[   0    0    0 ...  209  381   21]
 [   0    0    0 ...   53   36  271]
 [   0    0    0 ...    0    6  755]
 ...
 [   0    0    0 ...  215  243  190]
 [   0    0    0 ...  754  142 1040]
 [   0    0    0 ...    0    0   47]]

Training shape: (2198, 100)
Training type: <class 'list'>
Padded Training type: <class 'numpy.ndarray'>
```

```
# Display example of padded sentence
print("Padded sequence example:\n\n", training_sentences[3])
```

```
Padded sequence example:

 charger convers last minut major problem
```

```
# Output sentiment categories
print('Sentiment categories', df.sentiments.unique())
```

```
Sentiment categories [0 1]
```

```
df.groupby('sentiments').count()
```

sentiments	review	word_count	clean_review	polarity
0	1362	1362	1362	1362
1	1386	1386	1386	1386

B4. Categories of Sentiment

Summary of Sentiment

From our exploratory analysis we observed two categories of sentiment, 'negative' and 'positive' values represented by '0' and '1' respectively. The categories of sentiment will be used in the Keras Softmax module for analysis.

CODE

```
# Verify binary sentiment values take only 0 or 1 values
df.sentiments.unique()
```

```
array([0, 1], dtype=int64)
```

```
# Count total number of Positive and Negative sentiments present
print('Dataframe Sentiments \n', df.sentiments.value_counts())
```

```
Dataframe Sentiments
 1    1386
0    1362
Name: sentiments, dtype: int64
```

```
# Total Percentages of Positive and Negative sentiments
print('Merged Dataframe \n', df.sentiments.value_counts() / len(df))
```

```
Merged Dataframe
 1    0.504367
0    0.495633
Name: sentiments, dtype: float64
```

```
# Output sentiment categories
print('Sentiment categories', df.sentiments.unique())
```

```
Sentiment categories [0 1]
```

```
df.groupby('sentiments').count()
```

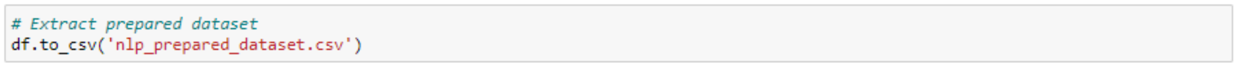
sentiments	review	word_count	clean_review	polarity
0	1362	1362	1362	1362
1	1386	1386	1386	1386

B5. Steps to Prepare the Data

The following steps were taken to prepare the data for use in the Neural Network:

1. Import the UCI Sentiment Labeled Sentences datasets using Pandas
2. Verify sentiments represented in binary 0 and 1 values
3. Verify text sequences do not contain special characters, emojis, etc.
4. Review metrics of sequence characteristics i.e., shape, size, type, etc.
5. Remove unnecessary hyperlink text and characters
6. Convert all sequences to lowercase characters
7. Removed both stop and rare words in sequences
8. Corrected grammar and spelling errors
9. Manipulated word bases using stems and lemmas
10. Created training and test set from split data
11. Created tokenizes sequences
12. Extracted dataset for review

B6. Prepared Dataset

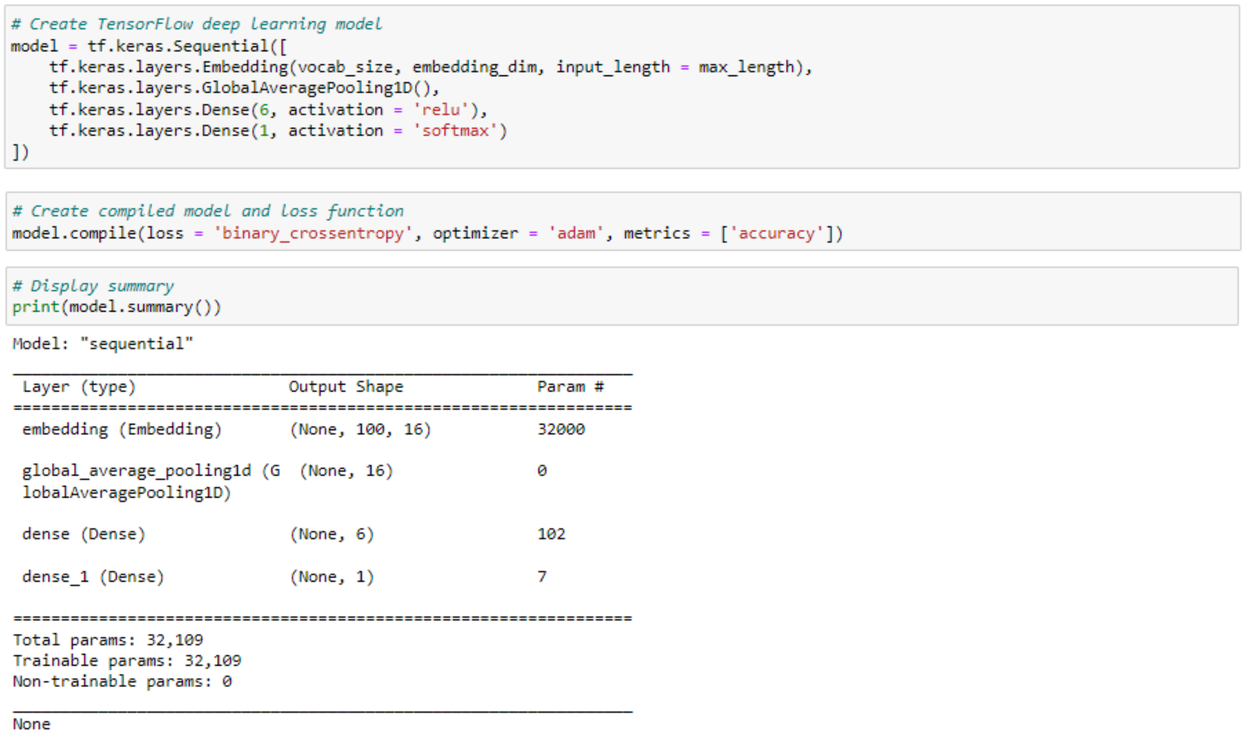


```
# Extract prepared dataset
df.to_csv('nlp_prepared_dataset.csv')
```

Part III: Network Architecture

C1. Model Summary

The following details output of the model summary from the TensorFlow function which aligns with the NLP Neural Network.



```
# Create TensorFlow 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')
])
```

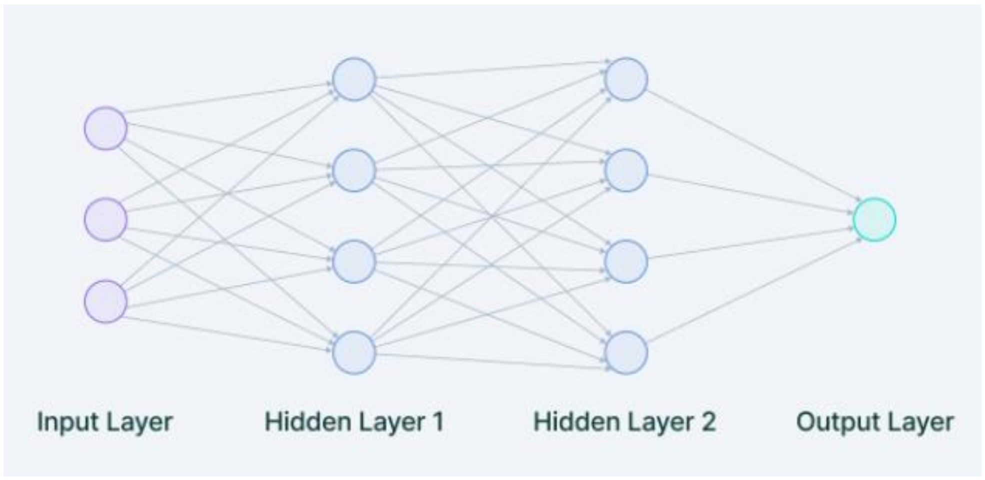
```
# Create compiled model and loss function
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
# Display summary
print(model.summary())
```

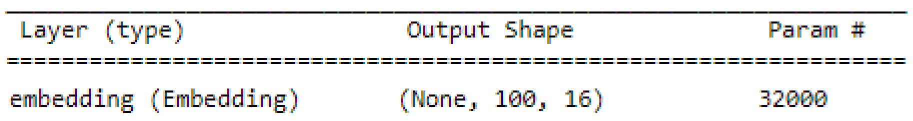
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 100, 16)           32000     
                                                                 
 global_average_pooling1d (G  (None, 16)               0         
 lobalAveragePooling1D)                                          
                                                                 
 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. Network Architecture

The Network Architecture of the Neural Network constitutes layers composed of multiple neurons stacked together in rows. Four layers are stacked together to create the multi-layer neural network as shown in the figure below (V7 2022).

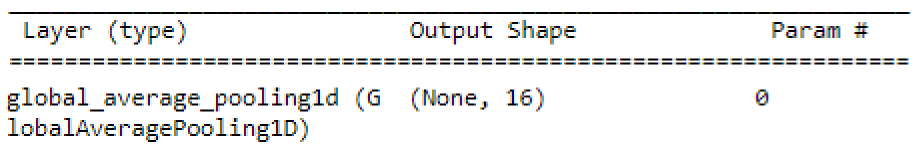


The first layer is the input layer and the only actual visible layer of the network stack. This input layer receives the UCI Sentiment datasets in .CSV format and passes the information without computation. From exploratory analysis the input layer describes a vocabulary size comprised of two thousand words, sixteen embedded dimensions, and max length of 100 words as seen in the following table.



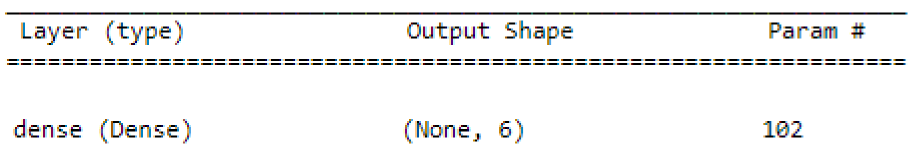
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
embedding (Embedding)       (None, 100, 16)           32000
```

The second layer is hidden and the beginning of the computational layers designed to extract features from the data. We can see evidence of computations from the second network layer in the following results.



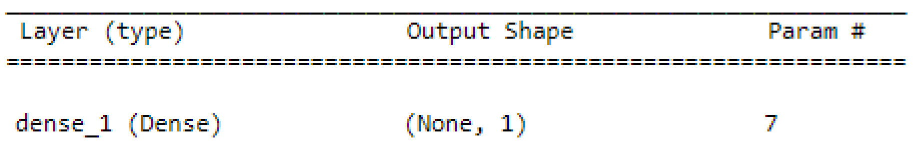
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
global_average_pooling1d (G  (None, 16)               0
lobalAveragePooling1D)
```

The third layer is again a hidden layer and the final computational layer in this network architecture. This stack represents a dense layer and describes six neurons and about one hundred parameters as seen in the following table.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================

 dense (Dense)              (None, 6)                 102
```

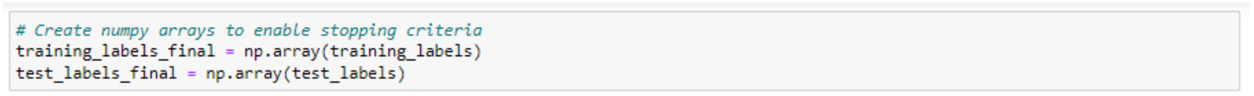
The fourth and final stack in the network architecture is the output layer. This layer takes the metrics from the previous two layers and creates a prediction resulting from the model's learnings. This layer produces the final results and is calculated using the imported Softmax function which describes a dense layer with seven parameters as seen in the following table.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================

 dense_1 (Dense)            (None, 1)                 7
```

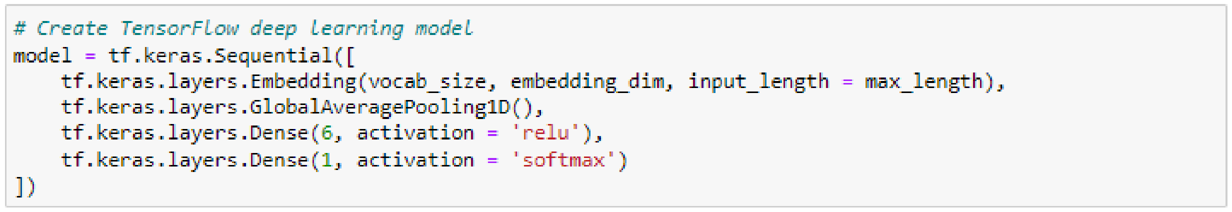
Overall we have calculated 32,000 parameters that are trainable and no parameters that are non-trainable.

C3. Hyperparameters



```
# Create numpy arrays to enable stopping criteria
training_labels_final = np.array(training_labels)
test_labels_final = np.array(test_labels)
```

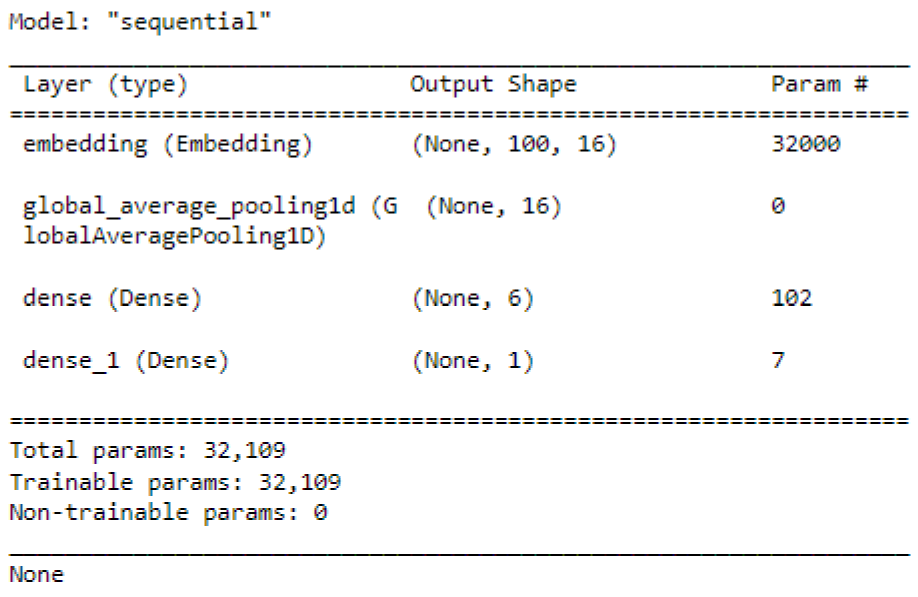
- **Activation Functions**



```
# Create TensorFlow 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')
])
```

The activation functions in the neural network are provided by the Keras library and include Relu and Softmax. These functions were chosen for efficiency in parsing larger dataset sequences.

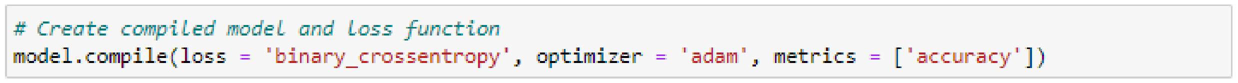
- **Number of nodes per layer**



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           32000

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 6)                 102

 dense_1 (Dense)             (None, 1)                 7

=================================================================
Total params: 32,109
Trainable params: 32,109
Non-trainable params: 0
_________________________________________________________________
None
```

In this Neural Network, we observe six nodes residing in the third layer and one node in the output layer. This amount of nodes satisfies the requirement for insightful metrics.

- **Loss Function**



```
# Create compiled model and loss function
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

The Loss Function invoked in the Neural Network is the binary_crossentropy module which is a loss function built on probability.

- **Optimizer**

```
# Create compiled model and loss function
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

The optimizer selected for the Neural Network is the Adam module. The Adam optimizer benefits from straightforward implementation as well as efficiency and low memory requirements (Brownlee 2021).

- **Stopping criteria**

```
# Import Earlystopping module from Keras library
import tensorflow
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

The Early Stopping module from the Keras library has been selected for the Neural Network. The stopping criteria is a function of EarlyStopping which activates when the monitored metric has stopped improving.

- **Evaluation metric**

```
# Calculate Accuracy
loss, accuracy = model.evaluate(padded, training_labels_final, verbose = 0)
print('Accuracy: %f' % (accuracy * 100))
```

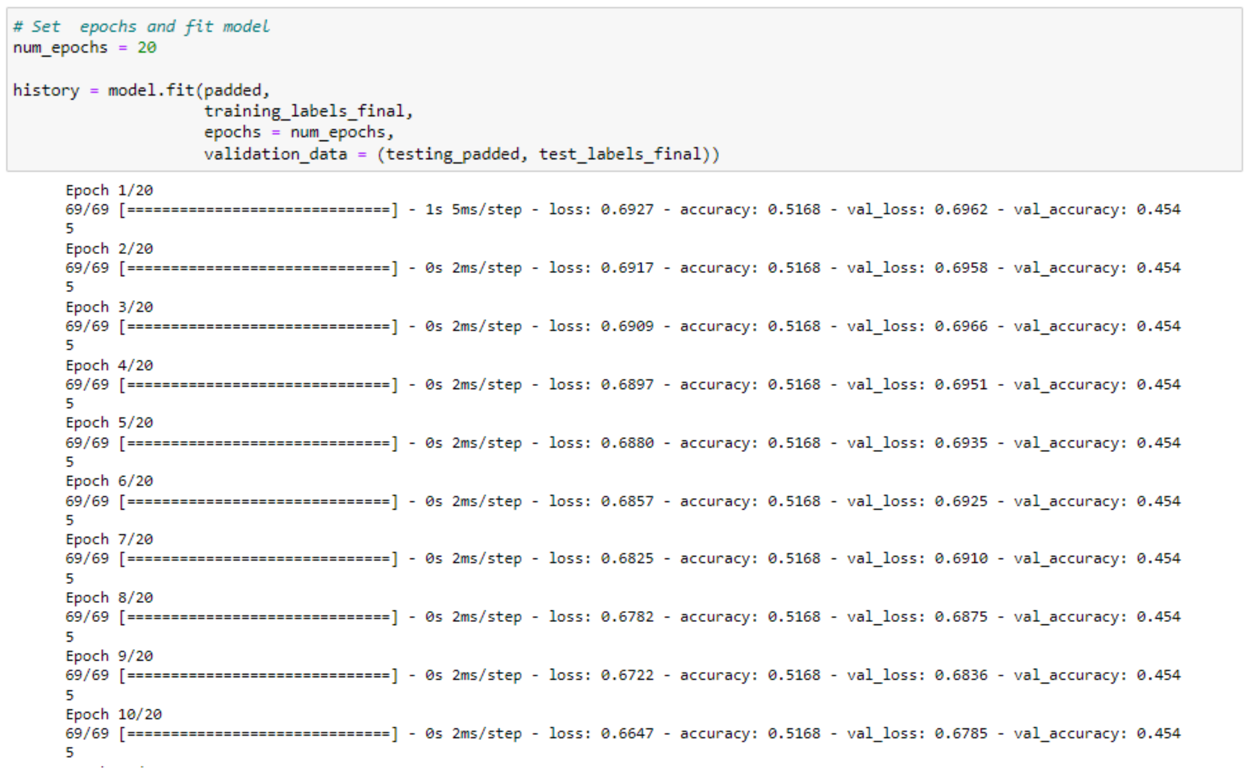
```
Accuracy: 51.683348
```

Our evaluation metric for the Neural Network is based on the accuracy metric which evaluates the model based on padded data and training labels.

Part IV: Model Evaluation

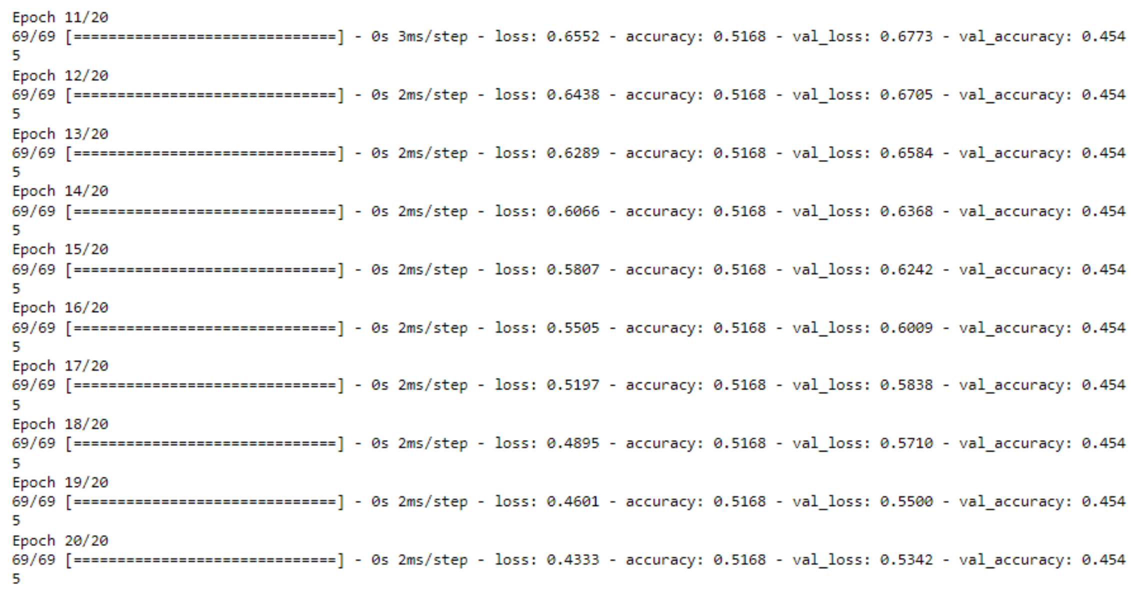
D1. Stopping Criteria

The Early Stopping module from the Keras library has been selected for the Neural Network. The stopping criteria is a function of EarlyStopping which activates when the monitored metric has stopped improving. The stopping criteria in our neural metric is met after 20 epochs after which accuracy increases become negligible.



```
# Set  epochs and fit model
num_epochs = 20

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

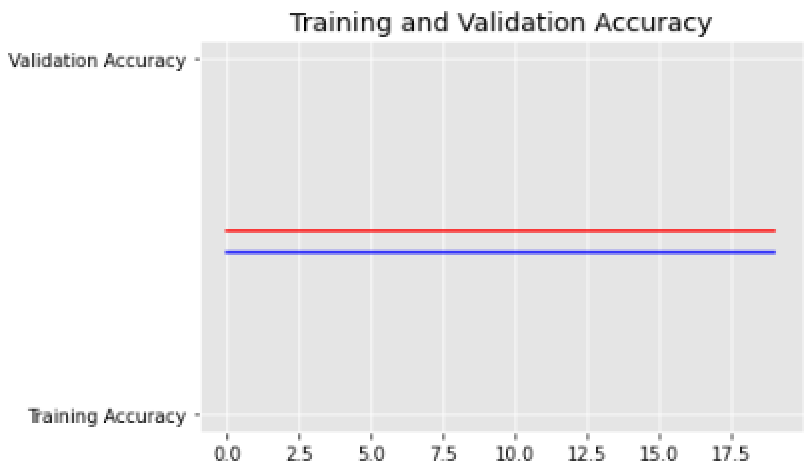
```
Epoch 1/20
69/69 [==============================] - 1s 5ms/step - loss: 0.6927 - accuracy: 0.5168 - val_loss: 0.6962 - val_accuracy: 0.454
5
Epoch 2/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6917 - accuracy: 0.5168 - val_loss: 0.6958 - val_accuracy: 0.454
5
Epoch 3/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6909 - accuracy: 0.5168 - val_loss: 0.6966 - val_accuracy: 0.454
5
Epoch 4/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6897 - accuracy: 0.5168 - val_loss: 0.6951 - val_accuracy: 0.454
5
Epoch 5/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6880 - accuracy: 0.5168 - val_loss: 0.6935 - val_accuracy: 0.454
5
Epoch 6/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6857 - accuracy: 0.5168 - val_loss: 0.6925 - val_accuracy: 0.454
5
Epoch 7/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6825 - accuracy: 0.5168 - val_loss: 0.6910 - val_accuracy: 0.454
5
Epoch 8/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6782 - accuracy: 0.5168 - val_loss: 0.6875 - val_accuracy: 0.454
5
Epoch 9/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6722 - accuracy: 0.5168 - val_loss: 0.6836 - val_accuracy: 0.454
5
Epoch 10/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6647 - accuracy: 0.5168 - val_loss: 0.6785 - val_accuracy: 0.454
5
```

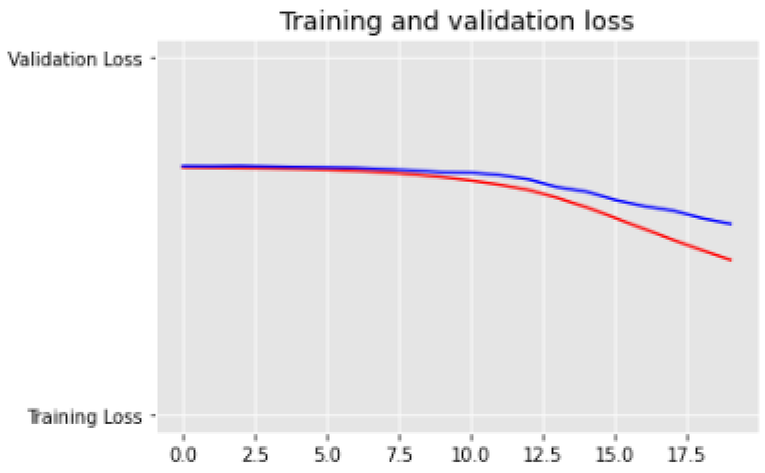


```
Epoch 11/20
69/69 [==============================] - 0s 3ms/step - loss: 0.6552 - accuracy: 0.5168 - val_loss: 0.6773 - val_accuracy: 0.454
5
Epoch 12/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6438 - accuracy: 0.5168 - val_loss: 0.6705 - val_accuracy: 0.454
5
Epoch 13/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6289 - accuracy: 0.5168 - val_loss: 0.6584 - val_accuracy: 0.454
5
Epoch 14/20
69/69 [==============================] - 0s 2ms/step - loss: 0.6066 - accuracy: 0.5168 - val_loss: 0.6368 - val_accuracy: 0.454
5
Epoch 15/20
69/69 [==============================] - 0s 2ms/step - loss: 0.5807 - accuracy: 0.5168 - val_loss: 0.6242 - val_accuracy: 0.454
5
Epoch 16/20
69/69 [==============================] - 0s 2ms/step - loss: 0.5505 - accuracy: 0.5168 - val_loss: 0.6009 - val_accuracy: 0.454
5
Epoch 17/20
69/69 [==============================] - 0s 2ms/step - loss: 0.5197 - accuracy: 0.5168 - val_loss: 0.5838 - val_accuracy: 0.454
5
Epoch 18/20
69/69 [==============================] - 0s 2ms/step - loss: 0.4895 - accuracy: 0.5168 - val_loss: 0.5710 - val_accuracy: 0.454
5
Epoch 19/20
69/69 [==============================] - 0s 2ms/step - loss: 0.4601 - accuracy: 0.5168 - val_loss: 0.5500 - val_accuracy: 0.454
5
Epoch 20/20
69/69 [==============================] - 0s 2ms/step - loss: 0.4333 - accuracy: 0.5168 - val_loss: 0.5342 - val_accuracy: 0.454
5
```

D2. Training Process

```
# Print Training and Validation accurary and training and validation loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs=range(len(acc))
plt.plot(epochs, acc, 'r', 'Training Accuracy')
plt.plot(epochs, val_acc, 'b', 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.figure()
plt.plot(epochs, loss, 'r', 'Training Loss')
plt.plot(epochs, val_loss, 'b', 'Validation Loss')
plt.title('Training and validation loss')
plt.figure()
```





```
# Calculate Accuracy
loss, accuracy = model.evaluate(padded, training_labels_final, verbose = 0)
print('Accuracy: %f' % (accuracy * 100))
```

```
Accuracy: 51.683348
```

D3. Fit

In calculating our Fitting, we first examine the sentiments of our dataset which taken on values of either 0 for negative or 1 for positive.

```
# Count total number of Positive and Negative sentiments present
print('Dataframe Sentiments \n', df.sentiments.value_counts())
```

```
Dataframe Sentiments
 1    1386
0    1362
Name: sentiments, dtype: int64
```

Next, we examine the size of our vocabulary which comes to 2717 words.

```
# Display number of unique words
print('Unique words: ' + str(len(unique)) + '.')
```

```
Unique words: 2717.
```

Then, we evaluate the stopping function of the neural network which results in a critical juncture at the initial epoch. It is as this point that we see the metric of **overfitting**. Although we have taken steps to clean the data including removing non-English characters, emojis, hyperlinks, etc the resulting overall accuracy is arguably low which is most likely a result of overfitting in the neural network model. It is recommended that we further review and manipulate the text sequences to prevent overfitting and increase overall accuracy by reducing the vocabulary list.

```
# Set  epochs and fit model
num_epochs = 20

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

```
Epoch 1/20
69/69 [==============================] - 1s 5ms/step - loss: 0.6927 - accuracy: 0.5168 - val_loss: 0.
6962 - val_accuracy: 0.4545
```

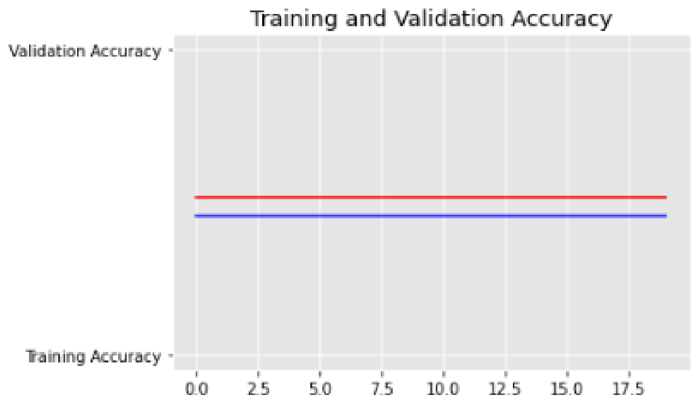
Finally, we set the activation functions including Relu and Softmax to increase the accuracy of our Neural Network model.

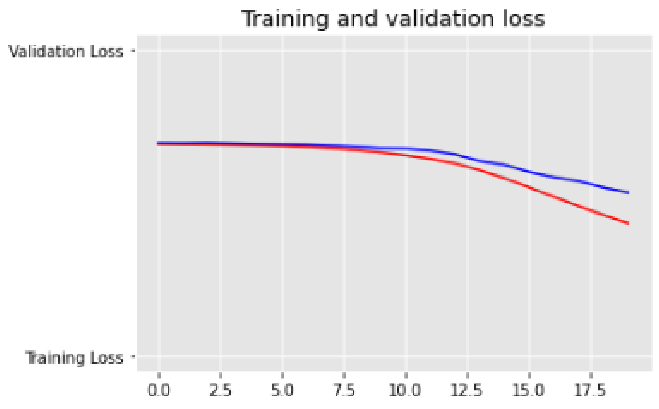
```
# Create TensorFlow deep learning model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length = max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'softmax')
])
```

D4. Predictive Accuracy

```
# Print Training and Validation accurary and training and validation loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs=range(len(acc))
plt.plot(epochs, acc, 'r', 'Training Accuracy')
plt.plot(epochs, val_acc, 'b', 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.figure()
plt.plot(epochs, loss, 'r', 'Training Loss')
plt.plot(epochs, val_loss, 'b', 'Validation Loss')
plt.title('Training and validation loss')
plt.figure()
```

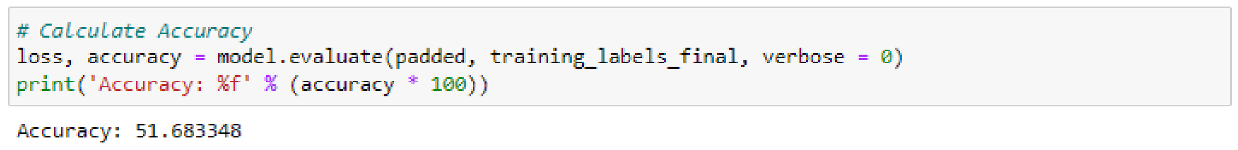
```
<Figure size 432x288 with 0 Axes>
```





The first metric of our trained network is a comparison of training and validation accuracy. The values are near parallel metrics which offer insights into the epochs. Having parallel values is an indicator that our sequences are derived from a small dataset with a batch having a large size. In the future we should move the mean and variance inputs to more appropriate values.

The plot showing training and validation loss gives us insight into the fitting of the model, and as training continues, we see degradation skewing the metrics apart which is an indicator that the data is overfit.



```
# Calculate Accuracy
loss, accuracy = model.evaluate(padded, training_labels_final, verbose = 0)
print('Accuracy: %f' % (accuracy * 100))
```

```
Accuracy: 51.683348
```

Predictive Accuracy in the Neural Network model is based on a binary classification scoring system. The accuracy store is based on the calculation:

$$\frac{(True\ Positives + True\ Negatives)}{(True\ Positive + True\ Negative + False\ Positve + False\ Negative}$$

Based on the observed value versus the predicted value we see a resulting accuracy score of 51%. Our Neural Network shows promise but we should aim to achieve a predictive accuracy score of 80% or greater. Based on observations of metrics we can assume that overfitting is the main culprit for the underwhelming accuracy score.

Part V: Summary and Recommendations

E. Code

This code for this assessment has been provided in the output of each respective section above.

F. Functionality

Our stated initial research question is:

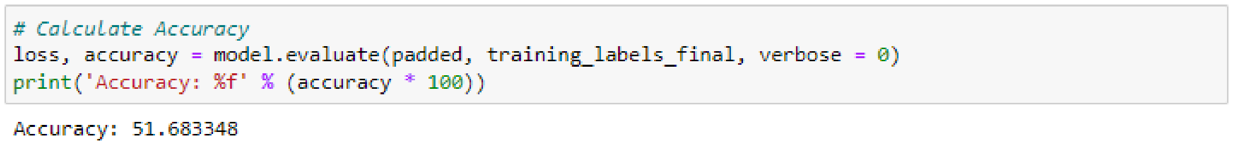
Is it possible to gain insightful sentiment metrics from the UCI Sentiment Labeled Sentences dataset containing customer reviews by using a Natural Language Processing Neural Network?

In this assignment, we accomplished building a Natural Language Processing Neural Network built on the TensorFlow and Keras libraries. To accomplish our goal, we inputted the UCI Sentiment Labeled Sentences Data Set composed of customer reviews from Amazon, IMDB, and Yelp.

Our predictive accuracy score of the Neural Network was built on a supervised learning model which is composed of four distinct network architecture layers. However, it appears that refinement is needed to increase the accuracy score, changes such as increasing the size of the inputted dataset or completing the model with improved activation functions.

G. Recommendations

The overall success of the Neural Network is defined by the observed predictive accuracy metric as shown in the figure below:



```
# Calculate Accuracy
loss, accuracy = model.evaluate(padded, training_labels_final, verbose = 0)
print('Accuracy: %f' % (accuracy * 100))

Accuracy: 51.683348
```

This score, unfortunately, does not instill confidence in the Neural Network model. The foundation of the Neural Network is solid, and will serve as a base foundation for future Neural Network and Deep learning models. However, it is our recommendation that we improve the model by increasing the size of the dataset inputs as the more data we have to train the more accurate our predictive results will become. An alternative to using a larger dataset to improve the results is to modify the libraries used in the analysis such as replacing with Relu and Softmax activation functions.

Part VI: Reporting

H. Reporting

Industry-relevant interactive development environment is provided in the attached Jupyter Notebook file:

D213_Task_2_Sentiment_Analysis.pdf

I. Sources for Third-Party Code

DataCamp. (n.d.). *Introduction to deep learning in python course*. DataCamp. Retrieved May 3, 2022, from https://www.datacamp.com/courses/introduction-to-deep-learning-in-python

DataCamp. (n.d.). *Deep learning in python*. DataCamp. Retrieved May 3, 2022, from https://www.datacamp.com/tracks/deep-learning-in-python

Keras tutorial: Deep Learning in python. DataCamp Community. (n.d.). Retrieved May 3, 2022, from https://www.datacamp.com/community/tutorials/deep-learning-python

Deep learning tutorial. DataCamp Community. (n.d.). Retrieved May 3, 2022, from https://www.datacamp.com/community/tutorials/tutorial-deep-learning-tutorial

DataCamp. (n.d.). *Introduction to deep learning with keras course*. DataCamp. Retrieved May 3, 2022, from https://www.datacamp.com/courses/introduction-to-deep-learning-with-keras

J. Sources

Kausar, A. (2019, August 14). *NLP in Tensorflow-all you need for a kickstart*. Medium. Retrieved April 30, 2022, from https://aqsakausar30.medium.com/nlp-in-tensorflow-all-you-need-for-a-kickstart-3293d7d2630e

What is tokenization: Tokenization in NLP. Analytics Vidhya. (2021, July 23). Retrieved May 2, 2022, from https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

Caner. (2020, April 3). *Padding for NLP*. Medium. Retrieved May 2, 2022, from https://medium.com/@canerkilinc/padding-for-nlp-7dd8598c916a

The Essential Guide to Neural Network Architectures. V7. (n.d.). Retrieved May 2, 2022, from https://www.v7labs.com/blog/neural-network-architectures-guide

Brownlee, J. (2021, January 12). *Gentle introduction to the adam optimization algorithm for deep learning*. Machine Learning Mastery. Retrieved May 3, 2022, from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/