Anthony Scampini

D213: Advanced Data Analytics Task 2

OCT-27-2024

Western Governors University

# A1: Research Question

For this assessment we are asking the research question "Can we use Natural Language Processing to predict whether a customer review is positive or negative." When there are many customer reviews for a product, manually determining the sentiment of each review is a very time-consuming process. By using the TensorFlow machine-learning framework, we can perform sentiment analysis to "understand the degree of positivity of a given piece of text." (Microsoft, n.d.). In doing so, customer reviews can be automatically categorized by sentiment to help decision makers better understand the public reception of their product without needing to manually sift through each individual review.

# A2: Objective or Goals

The goal of the analysis is to find whether we can accurately predict whether the sentiment of a customer review is either positive or negative. For training, we are provided with one-thousand reviews each from three distinct sources: Amazon, IMDB, and Yelp. Each review is labeled as the integers zero for negative and one for positive. A majority portion of this data is used for training the model while the remainder is used to test the model's accuracy when presented with unknown data. For the model, TensorFlow and the Keras API are used to train a sequential neural network which accepts an input parameter of a customer review and outputs either a one or a zero to classify the review as positive or negative. After hypertuning the model, the testing data is measured for accuracy to determine the model’s effectiveness. If the model proves to be accurate, it can be used by businesses to quickly sort through large numbers of reviews and classify them as either negative or positive. This will assist their marketing departments to help them better understand their customers and the public perception of their products.

# A3: Prescribed Network

The specific type of neural network that is being trained is an Artificial Neural Network (ANN) which “replicates how the human brain processes data in computer systems.” (Singh, 2021). This replication occurs through vectorization of tokens, which for this use-case are words, and running those vectors through layers of nodes which act as artificial neurons. Each node is weighted to represent the strength of its connection to another node, and the vector traverses the layers of nodes until a final output is achieved.

The Python library for TensorFlow provides a Sequential model which is appropriate for modeling customer reviews. For our purposes, this model is constructed with five layers:

1. The Embedding layer, which maps specific words to vectors.
2. GlobalAveragePooling1D layer, which averages the dimensions of the input.
3. Dense layer, 24 nodes of the Rectified Linear Unit function which introduces non-linearity to the deep learning model.
4. Dense layer, 24 nodes of the Softmax function, which introduces vectors of probabilities to the deep learning model.
5. Dense layer, 1 node of the Sigmoid function, which outputs either a zero for negative or one for positive.

In addition to the three activation functions, the loss function Binary Crossentropy and the optimizer function Adam are applied to the model. The model is evaluated by its accuracy and loss.

# B1: Data Exploration

Data exploration starts by first importing and combining the three datasets consisting of Amazon, Yelp and IMDB reviews. For each record in the files, a list item containing both the review value and the sentiment value is created.

filenames = ['amazon\_cells\_labelled.txt', 'imdb\_labelled.txt', 'yelp\_labelled.txt']  
lists = []  
for filename in filenames:  
 with open(filename) as data:  
 for line in data:  
 record = line.split('\t')  
 lists.append(record)  
print(lists[:5])

[['So there is no way for me to plug it in here in the US unless I go by a converter.', '0\n'], ['Good case, Excellent value.', '1\n'], ['Great for the jawbone.', '1\n'], ['Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!', '0\n'], ['The mic is great.', '1\n']]

In observing the first five records of the import, it is immediately apparent that each sentiment value of 0 or 1 is followed by \n which denotes a newline. To clean the sentiment column, the newlines are removed.

for record in lists:  
 record[1] = record[1].replace('\n','')  
print(lists[:5])

[['So there is no way for me to plug it in here in the US unless I go by a converter.', '0'], ['Good case, Excellent value.', '1'], ['Great for the jawbone.', '1'], ['Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!', '0'], ['The mic is great.', '1']]

Next, the data is analyzed to determine the presence of special characters. A list of every character that appears throughout the reviews is created.

char\_list = []  
for record in lists:  
 for words in record:  
 for chars in words:  
 if chars not in char\_list:  
 char\_list.append(chars)  
print(char\_list)

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

From the output above, it is observed that there is a combination of lowercase and uppercase alphanumeric characters, numbers, as well as an assortment of special characters. As part of the data pre-processing task for the model, the special characters are removed, and all alphabetic characters are converted to lowercase. Regular Expression, otherwise known as regex, is utilized in this process.

for record in lists:  
 record[0] = record[0].lower()  
 record[0] = re.sub('[^a-zA-Z0-9\s]', ' ', record[0])

char\_list = []  
for record in lists:  
 for words in record:  
 for chars in words:  
 if chars not in char\_list:  
 char\_list.append(chars)

print(char\_list)  
print(lists[:10])

['s', 'o', ' ', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', '0', 'd', 'x', '1', 'j', '4', '5', 'z', 'q', 'k', '7', '3', '6', '8', '2', '9']

[['so there is no way for me to plug it in here in the us unless i go by a converter ', '0'], ['good case excellent value ', '1'], ['great for the jawbone ', '1'], ['tied to charger for conversations lasting more than 45 minutes major problems ', '0'], ['the mic is great ', '1'], ['i have to jiggle the plug to get it to line up right to get decent volume ', '0'], ['if you have several dozen or several hundred contacts then imagine the fun of sending each of them one by one ', '0'], ['if you are razr owner you must have this ', '1'], ['needless to say i wasted my money ', '0'], ['what a waste of money and time ', '0']]

There are numerous instances of double and triple-spaces in the reviews, and each review ends with a trailing whitespace. All double and greater spaces from the reviews as well as the trailing whitespaces are removed.

for record in lists:  
 record[0] = re.sub(' +', ' ', record[0])  
 record[0] = record[0].strip()

print(lists[:10])

[['so there is no way for me to plug it in here in the us unless i go by a converter', '0'], ['good case excellent value', '1'], ['great for the jawbone', '1'], ['tied to charger for conversations lasting more than 45 minutes major problems', '0'], ['the mic is great', '1'], ['i have to jiggle the plug to get it to line up right to get decent volume', '0'], ['if you have several dozen or several hundred contacts then imagine the fun of sending each of them one by one', '0'], ['if you are razr owner you must have this', '1'], ['needless to say i wasted my money', '0'], ['what a waste of money and time', '0']]

To reduce the amount of processing power required by the neural network, stopwords are removed from the reviews. Stopwords are a set of commonly used words such as “the” or “and” that do not contribute towards the determination of sentiment for the review. A list of stopwords are imported from the NLTK library.

import nltk  
from nltk.corpus import stopwords  
nltk.download('stopwords')  
print(stopwords.words('english'))

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

Upon reviewing the list of stopwords, it is revealed that several of the stopwords contain an apostrophe. Since we already removed special characters from the reviews, the apostrophes need to be removed from the stopwords before they are applied.

cleaned\_stopwords = []  
for word in stopwords.words('english'):  
 cleaned\_stopwords.append(word.replace('\'',''))  
print(cleaned\_stopwords)

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'youre', 'youve', 'youll', 'youd', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'shes', 'her', 'hers', 'herself', 'it', 'its', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'thatll', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'dont', 'should', 'shouldve', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', 'arent', 'couldn', 'couldnt', 'didn', 'didnt', 'doesn', 'doesnt', 'hadn', 'hadnt', 'hasn', 'hasnt', 'haven', 'havent', 'isn', 'isnt', 'ma', 'mightn', 'mightnt', 'mustn', 'mustnt', 'needn', 'neednt', 'shan', 'shant', 'shouldn', 'shouldnt', 'wasn', 'wasnt', 'weren', 'werent', 'won', 'wont', 'wouldn', 'wouldnt']

Now that the list of stopwords is cleaned, each review is iterated through and any instance of one of the stopwords is removed.

for record in lists:  
 record[0] = ' '.join([word for word in record[0].split() if word not in cleaned\_stopwords])

print(lists[:10])

[['way plug us unless go converter', '0'], ['good case excellent value', '1'], ['great jawbone', '1'], ['tied charger conversations lasting 45 minutes major problems', '0'], ['mic great', '1'], ['jiggle plug get line right get decent volume', '0'], ['several dozen several hundred contacts imagine fun sending one one', '0'], ['razr owner must', '1'], ['needless say wasted money', '0'], ['waste money time', '0']]

Another method to improve computational performance is to remove one-character words. These single character words provide little insight into the sentiment of the review.

for record in lists:  
 record[0] = ' '.join([word for word in record[0].split() if len(word) > 1])

Now that the reviews are cleaned, they can be imported into a Pandas dataframe.

reviews = pd.DataFrame(lists, columns=['Review','Sentiment'])  
print(reviews.head(10).to\_string(index=False))

Review Sentiment  
way plug us unless go converter 0  
good case excellent value 1  
great jawbone 1  
tied charger conversations lasting 45 minutes major problems 0  
mic great 1  
jiggle plug get line right get decent volume 0  
several dozen several hundred contacts imagine fun sending one one 0  
razr owner must 1  
needless say wasted money 0  
waste money time 0

The dataframe is checked for null values, as well as confirming that there are only two binary options for Sentiment, with zero being negative and one being positive.

for (review, sentiment) in reviews.items():  
 print('Total missing values in variable %s is: ' % review + str(reviews[review].isnull().sum()))  
print(reviews.Sentiment.value\_counts())

Total missing values in variable Review is: 0  
Total missing values in variable Sentiment is: 0

0 1500  
1 1500  
Name: Sentiment, dtype: int64

To obtain the cleaned reviews vocabulary size, the Tokenizer method is called from the TensorFlow preprocessing library.

from tensorflow.keras.preprocessing.text import Tokenizer

tokenized = Tokenizer()  
tokenized.fit\_on\_texts(reviews.Review)  
print('Vocabulary size: ', len(tokenized.word\_index)+1)  
for key, val in tokenized.word\_index.items():  
 print(val, ':', key)

Vocabulary size: 5020

1 : good  
2 : great  
3 : movie  
4 : phone  
5 : film  
6 : one  
7 : food  
8 : like  
9 : place  
10 : time

Note that the list of vocabulary words above has been truncated at ten for brevity in this assessment. The full list of words is available on the attached script and HTML report. The vocabulary size is 5019, and one additional unit is added to account for a padding value, making the total 5020.

To find the optimal word embedding length, we find the fourth root of the vocabulary size.

vocab\_size = len(tokenized.word\_index)+1  
embed\_size = vocab\_size\*\*0.25  
print('Embedding size: ', embed\_size)

Embedding size: 8.417360532594135

For this model, the optimal embedding size is 8.

For padding, we next determine the maximum length of words in a single review. We can also obtain the minimum, median and mean length for analyzation.

review\_len = []  
for word\_len in reviews.Review:  
 review\_len.append(len(word\_len.split(" ")))

print('Maximum length of sequences: ', np.max(review\_len))  
print('Minimum length of sequences: ', np.min(review\_len))  
print('Median length of sequences: ', round(np.median(review\_len)))  
print('Mean length of sequences: ', round(np.mean(review\_len)))

Maximum length of sequences: 44  
Minimum length of sequences: 1  
Median length of sequences: 5  
Mean length of sequences: 6

From this output, we find the maximum length of sequences is 44, and that value will be used during the padding process.

See attached script for full code and annotations.

# B2: Tokenization

The goal of the tokenization process is to vectorize the individual words in each review so that they are represented numerically. To do this, from the TensorFlow preprocessing library the Tokenizer method is called. Each review is imported to the Tokenizer object using the fits\_on\_texts() method, and then each word is converted to a numerical value using the texts\_to\_sequences() method.

vocab\_tokenized = Tokenizer(num\_words=5019, oov\_token='OOV')  
vocab\_tokenized.fit\_on\_texts(reviews.Review)  
encoded\_reviews = vocab\_tokenized.texts\_to\_sequences(reviews.Review)

A before and after comparison of the reviews can be analyzed.

print('Before: ',lists[1][0])  
print('After: ',encoded\_reviews[1])  
print('')  
print('Before: ',lists[20][0])  
print('After: ',encoded\_reviews[20])  
print('')  
print('Before: ',lists[50][0])  
print('After: ',encoded\_reviews[50])  
print('')  
print('Before: ',lists[100][0])  
print('After: ',encoded\_reviews[100])

Before: good case excellent value  
After: [2, 71, 29, 409]

Before: went motorola website followed directions could get pair  
After: [117, 238, 745, 2119, 2120, 31, 28, 746]

Before: loud enough turn like  
After: [457, 54, 363, 9]

Before: integrated seamlessly motorola razr phone  
After: [2160, 1373, 238, 616, 5]

One observation that can be made about the vectorization output, based on the excerpt above, is that the word “motorola” is now represented by the integer value 238. Each word now has a corresponding integer value due to the tokenization process.

See attached script for full code and annotations.

# B3: Padding Process

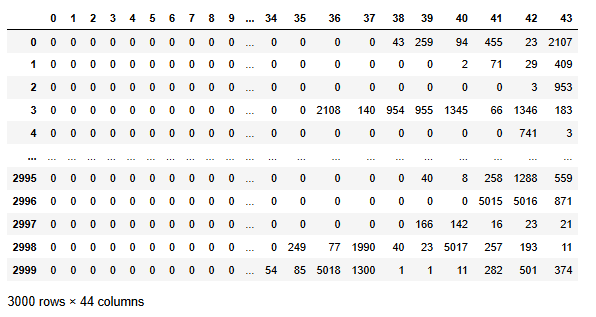
After the reviews are tokenized and converted into vectors, padding is used to ensure that all reviews maintain the same shape. We previously determined that the maximum length of sequences, or words, in a review is forty-four.

print('Maximum length of sequences: ', np.max(review\_len))  
Maximum length of sequences: 44

Each vectorized review is padded to reach a length of 44 sequences. The padding inserts 0 values in the beginning of each sequence until the sequence has reached a length of 44. Thus, a review with thirty sequences will begin with fourteen zeros to ensure that each vector is of the same size.

The padding is accomplished using the pad\_sequences() method from the TensorFlow preprocessing library. The resulting padded reviews are migrated to a dataframe for analysis.

padded\_reviews = pad\_sequences(encoded\_reviews, maxlen=44)  
padded\_reviews = pd.DataFrame(padded\_reviews)  
padded\_reviews



The above screenshot demonstrates the encoded and padded reviews, with forty-four sequences to accommodate the longest review. Each review with less than forty-four sequences are padded with zeros so that each review is of equal shape.

See attached script for full code and annotations.

# B4: Categories of Sentiment

We know that there are exactly two categories of sentiment provided by the datasets. The choice is binary, with zero being a negative review and one being a positive review.

print(reviews.Sentiment.value\_counts())  
0 1500  
1 1500  
Name: Sentiment, dtype: int64

Because there are only two levels for the output, the final dense layer of the network uses the activation function Sigmoid. The Sigmoid function is “usually used in output layer of a binary classification, where result is either 0 or 1.” (Geeks for Geeks, n.d.).

# B5: Steps to Prepare the Data

With the dataset being prepared for the neural network, the train\_test\_split() method from the SciKit-Learn Model Selection library is called to split the data into training and testing sets. The model uses 80% of the data, or 2400 reviews, to train. The model is then tested for accuracy and loss using the testing set which consists of the remaining 600 reviews. Because the sentiment values are currently stored in the dataframe as objects, they are converted to integers so that they can be consumed by the model.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(padded\_reviews, reviews.Sentiment.astype(int), test\_size=0.2)

Four datasets are created from the train\_test\_split() method, each containing an index and a value:

* X\_train: The reviews of the training data.
* X\_test: The reviews of the testing data.
* y\_train: The sentiment values of the training data.
* y\_test: The sentiment values of the testing data.

See attached script for full code and annotations.

# B6: Prepared Data Set

The prepared datasets are output to comma delimited files and attached to this assessment with the following filenames:

1. d213\_task2\_Scampini\_Xtrain.csv: Training data, reviews
2. d213\_task2\_Scampini\_Xtest.csv: Testing data, reviews
3. d213\_task2\_Scampini\_ytrain.csv: Training data, sentiment values
4. d213\_task2\_Scampini\_ytest.csv: Testing data, sentiment values

The to.csv() method is called to output the datasets to CSV files that can be shared.

X\_train.to\_csv('d213\_task2\_Scampini\_Xtrain.csv', index=False)  
X\_test.to\_csv('d213\_task2\_Scampini\_Xtest.csv', index=False)  
y\_train.to\_csv('d213\_task2\_Scampini\_ytrain.csv', index=False)  
y\_test.to\_csv('d213\_task2\_Scampini\_ytest.csv', index=False)

See attached script for full code and annotations.

# C1: Model Summary

Two libraries are required for training of the model:

* TensorFlow’s EarlyStopping method from the Callbacks library.
* TensorFlow’s Sequential method from the Models library.

In addition to those two libraries, TensorFlow itself is imported as the variable tf.

import tensorflow as tf  
from tensorflow.keras.callbacks import EarlyStopping  
from tensorflow.keras.models import Sequential

model = tf.keras.Sequential([  
 tf.keras.layers.Embedding(vocab\_size, round(embed\_size)),  
 tf.keras.layers.GlobalAveragePooling1D(),  
 tf.keras.layers.Dense(24, activation='relu'),  
 tf.keras.layers.Dense(24, activation='softmax'),  
 tf.keras.layers.Dense(1, activation='sigmoid')  
])

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])  
history = model.fit(X\_train, y\_train, epochs=30, batch\_size=48, callbacks=tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2), verbose=True)  
model.summary()

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

See attached script for full code and annotations.

# C2: Network Architecture

The summary() method provides information about the model used to analyze its architecture.

model.summary()

A screenshot of a computer

Description automatically generated

The model is built to support five layers:

1. The embedding layer, which accepts input dimensions equal to the total vocabulary size and outputs dimensions equal to the embedding size, which is equal to the fourth root of the vocabulary size. In this model, the values are 5020 and 8.
2. The pooling layer, which outputs the averages of the dimensions using the GlobalAveragePooling1D() method. This provides a consistent vector shape for the upcoming dense layers.
3. A dense layer consisting of 24 nodes of the Rectified Linear Unit activation function.
4. A dense layer consisting of 24 nodes of the Softmax function.
5. A dense layer consisting of a single Sigmoid node.

From the model summary output, we find that the model consists of:

* 41,001 trainable parameters.
  + 40,160 are in the embedding layer.
  + 216 are in the first dense layer for the Rectified Linear Unit activation function.
  + 600 are in the second dense layer for the Softmax function.
  + 25 are in the third dense layer for the Sigmoid function.
* 82,004 optimizer parameters.

C3: Hyperparameters

When building a natural language processing model using TensorFlow, several hyperparameters need to be defined. The final selection of parameters is determined by analyzing the data during preprocessing, a consideration of the context of the research question in relation to the available data, and the metrics of the tokenized dataset.

Three activation functions are selected for the dense layers of the model.

* Relu, or Rectified Linear Unit. This is a mathematical function that provides non-linearity to the data which helps the neural network better recognize patterns.
* Softmax, another mathematical function that converts output scores into probabilities.
* Sigmoid, the final dense layer of the model, which outputs the final binary prediction of whether a review is positive or negative.

The number of nodes per dense layer is acquired through experimentation. Once the model is built, the number of nodes within both the Relu and Softmax layers are experimented with until the values that produce the best accuracy are discovered. For this model, it was discovered that 24 nodes for both the Relu and Softmax dense layers consistently produce the most accurate model. The final sigmoid layer contains only one node as it is the output layer. The first layer, the embedding layer, contains as many nodes as the vocabulary size plus one, which for this model is 5,020 nodes. There is also a pooling layer that comes directly after the embedding layer, which does not handle connectivity or weights, instead performing an averaging operation that provides the dense layers with fixed length input vectors.

The loss function hyperparameter chosen for this model is binary crossentropy. The reason this loss function is selected is determined by the nature of the classification desired from the output. Because we are looking for an output of two classes, which is a binary choice, binary crossentrpoy is best suited for the classification problem. If for example the output consisted of many classes, categorical crossentropy would be better suited.

The optimizer function is what helps the model determine the optimal weights between the network nodes. For this model, the Adam optimizer function was applied. The Adam optimizer applies two mathematical functions: the Adaptive Gradient Algorithm and Root Mean Square Propagation. These functions update the network’s weights and maintains a fixed learning rate during training.

A stopping criteria is applied to the model to prevent overfitting. During training, the stopping criteria looks for a pattern to determine at which epoch the model is no longer benefitting from additional training. This is called using the TensorFlow method EarlyStopping(). Two parameters are provided to the method, the metric to evaluate and how many consecutive degradations that metric needs to see before halting training. For this model, we considered the loss metric with a patience of two, meaning that for each epoch if the loss increased in value two times in a row, the stopping criteria would be met and the model would not benefit from additional training.

The evaluation metric selected for this model is the accuracy, as it best fits the research question of predicting whether a review is negative or positive. For each review, we want to know how accurate the model is at making this prediction. Defining the evaluation metric as accuracy ensures that the performance of the model is being judged by how accurate it is in classifying the data.

D1: Stopping Criteria

While a stopping criteria was provided for the model, it proved to be most useful during hyperparameter tuning to find the model with the best accuracy. The stopping criteria saved time during this experimentation phase by halting the training early instead of iterating through every epoch. Models that stopped early often returned low accuracy and the hyperparameters would be adjusted seeking improvement.

For this model, the following stopping criteria was defined using the TensorFlow method EarlyStopping():

tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2, restore\_best\_weights=True)

The model is evaluating the loss metric of each epoch. During training, the loss value should decrease with each iteration. Because the patience is set at 2, if the loss increases twice in a row, training stops. For our final model, we can complete 30 epochs without reaching the stopping criteria, and further experimentation found that increasing the number of epochs to a point in which the stopping criteria is met results in overfitting and reduced accuracy. The screenshot below shows the training epochs of the model:

A screen shot of a computer

Description automatically generated

The final epoch is 30 with an accuracy of .97 and a loss of .20.

See attached script for full code and annotations.

# D2: Fitness

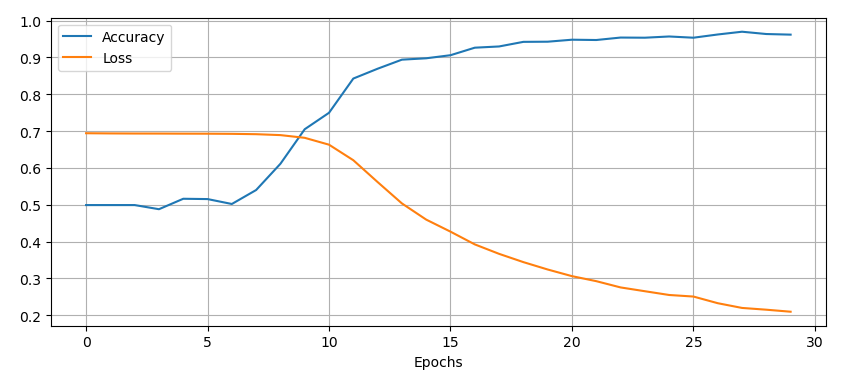
The fitness of the model is addressed during hyperparameter tuning to achieve the highest accuracy score during training. To mitigate overfitting, stopping criteria was applied to the model which halted training when the loss metric increased two epochs in a row. For the final model, 30 epochs were used and the stopping criteria was not met. Adding additional epochs to the model to the point that the stopping criteria would be met resulted in a decrease in overall accuracy of the model. Through this, we determined that thirty epochs produced a model that is optimally fit and does not show significant signs of underfitting or overfitting.

# D3: Training Process

The training process is evaluated at each epoch iteration using the accuracy and loss values as a metric. The accuracy defines how accurate the model was at predicting the sentiment of the review, while the loss metric defines the uncertainty of the prediction. Typically, an increase in accuracy results in a decrease in loss, and for each epoch iteration we expect to see just that. Once the model begins showing a decrease in accuracy or an increase in loss for consecutive epochs, an assumption can be made that overfitting is occurring.

The model’s training process can be visualized using line charts, with the epochs as the X axis and the accuracy/loss as the Y axis.

plt.figure(figsize = [10,4])  
plt.plot(history.history['accuracy'],label='Accuracy')  
plt.plot(history.history['loss'],label='Loss')  
plt.xlabel('Epochs')  
plt.grid()  
plt.legend()  
plt.show()



From the chart above, we observe that the accuracy of the model sees significant increase between the sixth and thirteenth epochs, while the loss of the model sees a steady decrease throughout the entirety of the thirty epochs. Improvements for both taper off at around 30 epochs.

See attached script for full code and annotations.

# D4: Predictive Accuracy

Prior to training the model, the original dataset of three thousand reviews was split into training and testing data with an 80/20 split, meaning that 80% of the reviews were used in training and the remaining 20% of the reviews are to be used in testing the model’s accuracy.

The testing data is fed into the trained model using the evaluate() method from the TensorFlow models library. This assesses the performance of the model when evaluating previously unseen data, which in this case is the testing data split off from the original dataset.

result = model.evaluate(X\_test,y\_test)



In testing the model, we find that the model can accurately predict the sentiment of a review 82% of the time. The loss value of .41 indicates that there is a level of uncertainty in the data that is higher than what was achieved during training.

# E: Code

Throughout this assessment, portions of the code were used to demonstrate the preprocessing phase and the training of the model. The full and complete code is attached to this assessment as the file “D213\_Task2\_Scampini\_Code.ipynb”, which was created using a Jupyter Notebook.

Below is the full code used in the creation and saving of the natural language processing model:

import re  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

import nltk  
from nltk.corpus import stopwords

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

import tensorflow as tf  
from tensorflow.keras.callbacks import EarlyStopping  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
from tensorflow.keras.models import Sequential

filenames = ['amazon\_cells\_labelled.txt', 'imdb\_labelled.txt', 'yelp\_labelled.txt']  
lists = []

for filename in filenames:  
 with open(filename) as data:  
 for line in data:  
 record = line.split('\t')  
 lists.append(record)

print(lists[:5])

for record in lists:  
 record[1] = record[1].replace('\n','')

print(lists[:5])

char\_list = []  
for record in lists:  
 for words in record:  
 for chars in words:  
 if chars not in char\_list:  
 char\_list.append(chars)

print(char\_list)

for record in lists:  
 record[0] = record[0].lower()  
 record[0] = re.sub('[^a-zA-Z0-9\s]', ' ', record[0])

char\_list = []  
for record in lists:  
 for words in record:  
 for chars in words:  
 if chars not in char\_list:  
 char\_list.append(chars)

print(char\_list)

print(lists[:10])

for record in lists:  
 record[0] = re.sub(' +', ' ', record[0])  
 record[0] = record[0].strip()

print(lists[:10])

nltk.download('stopwords')  
print(stopwords.words('english'))

cleaned\_stopwords = []

for word in stopwords.words('english'):  
 cleaned\_stopwords.append(word.replace('\'',''))

print(cleaned\_stopwords)

for record in lists:  
 record[0] = ' '.join([word for word in record[0].split() if word not in cleaned\_stopwords])

print(lists[:10])

for record in lists:  
 record[0] = ' '.join([word for word in record[0].split() if len(word) > 1])

for (review, sentiment) in reviews.items():  
 print('Total missing values in variable %s is: ' % review + str(reviews[review].isnull().sum()))

print(reviews.Sentiment.value\_counts())

tokenized = Tokenizer()  
tokenized.fit\_on\_texts(reviews.Review)  
print('Vocabulary size: ', len(tokenized.word\_index)+1)  
for key, val in tokenized.word\_index.items():  
 print(val, ':', key)

review\_len = []

for word\_len in reviews.Review:  
 review\_len.append(len(word\_len.split(" ")))

print('Maximum length of sequences: ', np.max(review\_len))  
print('Minimum length of sequences: ', np.min(review\_len))  
print('Median length of sequences: ', round(np.median(review\_len)))  
print('Mean length of sequences: ', round(np.mean(review\_len)))

vocab\_size = len(tokenized.word\_index)+1  
embed\_size = vocab\_size\*\*0.25  
print('Embedding size: ', embed\_size)

vocab\_tokenized = Tokenizer(num\_words=5019, oov\_token='OOV')  
vocab\_tokenized.fit\_on\_texts(reviews.Review)  
encoded\_reviews = vocab\_tokenized.texts\_to\_sequences(reviews.Review)

print('Before: ',lists[1][0])  
print('After: ',encoded\_reviews[1])  
print('')  
print('Before: ',lists[20][0])  
print('After: ',encoded\_reviews[20])  
print('')  
print('Before: ',lists[50][0])  
print('After: ',encoded\_reviews[50])  
print('')  
print('Before: ',lists[100][0])  
print('After: ',encoded\_reviews[100])

padded\_reviews = pad\_sequences(encoded\_reviews, maxlen=44)  
padded\_reviews = pd.DataFrame(padded\_reviews)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(padded\_reviews, reviews.Sentiment.astype(int), test\_size=0.2)

X\_train.to\_csv('d213\_task2\_Scampini\_Xtrain.csv', index=False)  
X\_test.to\_csv('d213\_task2\_Scampini\_Xtest.csv', index=False)  
y\_train.to\_csv('d213\_task2\_Scampini\_ytrain.csv', index=False)  
y\_test.to\_csv('d213\_task2\_Scampini\_ytest.csv', index=False)

model = tf.keras.Sequential([  
 tf.keras.layers.Embedding(vocab\_size, round(embed\_size)),  
 tf.keras.layers.GlobalAveragePooling1D(),  
 tf.keras.layers.Dense(24, activation='relu'),  
 tf.keras.layers.Dense(24, activation='softmax'),  
 tf.keras.layers.Dense(1, activation='sigmoid')  
])

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(X\_train, y\_train, epochs=30, batch\_size=48,  
 callbacks=tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2, restore\_best\_weights=True), verbose=True)

model.summary()

plt.figure(figsize = [10,4])  
plt.plot(history.history['accuracy'],label='Accuracy')  
plt.plot(history.history['loss'],label='Loss')  
plt.xlabel('Epochs')  
plt.grid()  
plt.legend()  
plt.show()

result = model.evaluate(X\_test,y\_test)

model.save("D213\_Task2\_Scampini\_Model.keras")

See attached script for annotations. See attached file “D213\_Task2\_Scampini\_Model.keras” for the saved model.

# F: Functionality

The network architecture of this Sequential neural network consists of an embedding layer, a pooling layer, and three dense layers using a combination of the Relu, Softmax, and Sigmoid activation functions. Through hyperparameter tuning we determined that 24 nodes of both the Relu and Softmax functions produce the most accurate results. In the end, we have a model that can accurately predict the sentiment of a review about 82% of the time.

The research question for this assessment is to answer whether we can use natural language processing to predict whether a customer review is positive or negative. An accuracy rate of 82% indicates that the NLP model can predict the sentiment of a review about 4 out of 5 times. This is certainly better than a coin-flip, but in my opinion not good enough to be authoritative for business decision-making purposes. While this model can be used to give decision makers a general overview of customer sentiment, a stronger model that has an accuracy in the mid-to-upper nineties would be preferrable for basing strategic decisions on.

# G: Recommendations

Because the model has an 82% accuracy when determining the sentiment of a review, the model can provide a general overview to decision makers relating to customer sentiment of a product based on its reviews. This general overview incorrectly predicts the sentiment 1 out of 5 times, so the output should not be considered authoritative and other techniques should be incorporated to define a business strategy. Ideally, an accuracy rating of 95% or higher should be achieved before considering the output useful in decision making.

It’s possible that training the data with a larger dataset from more specific sources relating to the type of review we are interested in could produce greater accuracy. Currently, the model was trained on one thousand reviews from three distinct sources: Amazon, Yelp, and IMDB. Considering that each source contains a different type of review: Amazon is product reviews, Yelp is business reviews, and IMDB is movie reviews, it’s possible that a more accurate model can be trained on a specific type of review only. For example, if we are interested in obtaining accurate sentiment for product reviews, it doesn’t make a lot of sense to include movie reviews in the training process. Having a large list of product reviews from several different commerce sites to train with will likely produce a model that is better at predicting the sentiment of product reviews.

# H: Reporting

Attached to this assessment is an exported HTML file showing the entirety of the code, annotations and output. The file was generated using Jupyter Notebook.

See attached file “D213\_Task2\_Scampini\_Report.html”

# I: Sources for Third-Party Code

Kotzias, D. (2015). “Sentiment Labelled Sentences.”  
 <https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences>

Elleh, F. (n.d.). “Welcome to D213 Advanced Data Analytics – Task 2”  
 <https://westerngovernorsuniversity.sharepoint.com/:p:/r/sites/DataScienceTeam/Shared%20Documents/Graduate%20Team/D213/Student%20Facing%20Resources/D213%20Task%201%20Cohort%20Webinar%20PPT.pptx>

Elleh, F. (n.d.). “Data Preprocessing Python Task 2”  
 <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=8639374a-964b-4ae9-b33b-b1210052c07d>

# J: Sources

Microsoft. (n.d.) “Introduction to Natural Language Processing with TensorFlow.” <https://learn.microsoft.com/en-us/training/modules/intro-natural-language-processing-tensorflow/1-introduction>

Singh, G. (2021). “Introduction to Artificial Neural Networks.” <https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/>

Geeks for Geeks. (n.d.) “Activation Functions in Neural Networks.” <https://www.geeksforgeeks.org/activation-functions-neural-networks/>