Matthew Blessing
*Student ID: **000507424***
NLM2 - Task 2, D213 - Advanced Data Analytics

NLM 2 - Performance Assessment Task 2: Sentiment Analysis Using Neural Networks

Part I: Research Question

A1.
In this analysis we will explore if it is possible to utilize a neural network model, coupled with natural language processing (NLP) techniques, to analyze the sentiment of online customer reviews.

A2.
The goals for this analysis will be to create a neural network model that can utilize NLP techniques to analyze online customer reviews and determine their sentiment, in addition to analyzing the accuracy and usefulness of the predictions of the model.

A3.
For this analysis, we have identified Keras as a neural network that will be capable of performing text classification that will be able to be trained and make predictions in our dataset of customer reviews. Keras is a deep learning, neural network API that is great for experimentation, allowing users to experiment with neural networks quickly, without needing to implement every layer and piece of code (Real Python, n.d.).

Part II: Data Preparation

B1.
To begin our analysis, we will first import our three .txt files containing customer reviews and concatenating those three sets of reviews into one data frame. We will also add a 'source' column, so that we can identify which website the reviews originated:

```
In [28]: #imports datasets, concatenates into a single data frame and separates into columns for sentence, label, and source

         data = {'yelp': 'yelp_labelled.txt',
                 'amazon': 'amazon_cells_labelled.txt',
                 'imdb':   'imdb_labelled.txt'}

         list = []
         for source, data in data.items():
             df = pd.read_csv(data, names=['sentence', 'label'], sep='\t')
             df['source'] = source
             list.append(df)

         df = pd.concat(list)
```

With all of our data in a single data frame, we can now explore the information contained within.

We can utilize Pandas .info() function to get a quick overview of our data, including variables, data types, and number of observations:

```
In [3]: #returns data types & counts for our data frame
        df.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 2748 entries, 0 to 747
        Data columns (total 3 columns):
         #   Column    Non-Null Count  Dtype
        ---  ------    --------------  -----
         0   sentence  2748 non-null   object
         1   label     2748 non-null   int64
         2   source    2748 non-null   object
        dtypes: int64(1), object(2)
        memory usage: 85.9+ KB
```

We can see we have 2,748 observations and 3 columns, 'sentence', 'label', and 'source', with data types object, int64, and object, respectively, which makes sense given the data.

We next want to determine if we have any duplicate rows, remove any duplicate rows, and check for any missing values:

```
In [5]: #checks for duplicate rows
        df.duplicated().sum()

Out[5]: 17

In [7]: #drops the 17 duplicate rows from our dataframe
        df.drop_duplicates(inplace=True)

In [9]: df.duplicated().sum()

Out[9]: 0

In [6]: #checks for null values
        df.isnull().sum()

Out[6]: sentence    0
        label       0
        source      0
        dtype: int64
```

As we can see, we had 17 duplicate rows in the dataset that we have dropped from the data frame. We have also checked for null values, but there are none.

We then utilize .value_counts() to see how many values exist for our three different sources, and counts for our labels, 1 and 0:

```
In [62]: df.source.value_counts()

Out[62]: yelp      1000
         amazon    1000
         imdb       748
         Name: source, dtype: int64

In [63]: df.label.value_counts()

Out[63]: 1    1386
         0    1362
         Name: label, dtype: int64
```

Our .value_counts() function informs us that:

- There are 1,000 rows where 'yelp' is the source
- There are 1,000 rows where 'amazon' is the source
- There 1,386 rows where the 'label' is 1 (positive review)
- There are 1,362 rows where the 'label' is 0 (negative review)

Next, we want to determine if there are any unusual characters in our dataset, outside of alphanumeric characters. We will utilize a .str.findall() function on the 'sentence' column to count the number of non-alphanumeric characters and sum those into a separate 'non_alphanumeric' column in the data frame. Note that it will not count multiple occurrences of the same non-alphanumeric character in the sentence, but will count each non-alphanumeric character once. This will inform us whether our reviews contain any non-alphanumeric characters.

```
In [66]: #creates a column in the df called 'non_alphanumeric' that counts the sum of non-alphanumeric characters in the 'sentence' column
         df['non_alphanumeric'] = df['sentence'].str.findall(r'[^a-zA-Z0-9 ]').str.len()

In [67]: df.head()
```

Out[67]:

	sentence	label	source	non_alphanumeric
0	Wow... Loved this place.	1	yelp	4
1	Crust is not good.	0	yelp	1
2	Not tasty and the texture was just nasty.	0	yelp	1
3	Stopped by during the late May bank holiday of...	1	yelp	1
4	The selection on the menu was great and so wer...	1	yelp	1

```
In [68]: #returns sum of all non-alphanumeric chars in the dataset
         df.non_alphanumeric.sum()

Out[68]: 6970
```

We can see that we have created our new column, 'non-alphanumeric', and counted the unique non-alphanumeric characters in each row's 'sentence' value. We then sum the total for the 'non-alphanumeric' column, which informs us the total number of non-alphanumeric characters in our dataset is 6,970. We will address and remove these characters later before building our model.

We can also see the statistics of our 'sentence' column by utilizing the .describe() function on the string length of the 'sentence' column:

```
In [69]: #desribes the statistics of the sentence column
         df['sentence'].str.len().describe()

Out[69]: count    2748.000000
         mean       71.528384
         std       201.987266
         min         7.000000
         25%        32.000000
         50%        55.000000
         75%        87.000000
         max      7944.000000
         Name: sentence, dtype: float64
```

This shows us that the average sentence length is 71.5 characters, along with the smallest entry in the 'sentence' column being 7 characters, and maximum being a rather lengthy 7,944 characters.

Next, we will want to determine the vocabulary size, or number of unique words used in the entirety of our 'sentence' values.

To begin, we will create a loop, utilizing the 'digits' and 'punctuation' libraries from 'string' in order to remove all numeric and punctuation characters from our three text files (Vocabulary Size in CSV File, 2013):

```
In [72]: #creates loops to iterate through each txt file and extracts all words, removing digits and punctuation

         from string import digits, punctuation
         remove_set = digits + punctuation
         with open("yelp_labelled.txt") as fin:
             yelp_vocab = {yelp_vocab.lower().strip(remove_set) for line in fin
         for yelp_vocab in line.rsplit(",",1)[0].split()}


         remove_set = digits + punctuation
         with open("amazon_cells_labelled.txt") as fin:
             amzn_vocab = {amzn_vocab.lower().strip(remove_set) for line in fin
         for amzn_vocab in line.rsplit(",",1)[0].split()}


         remove_set = digits + punctuation
         with open("imdb_labelled.txt") as fin:
             imdb_vocab = {imdb_vocab.lower().strip(remove_set) for line in fin
         for imdb_vocab in line.rsplit(",",1)[0].split()}
```

With all of the numeric and punctuation characters removed from the text files, we can then place those vocabulary lists into a new data frame called 'vocab', check for, and drop, duplicate values, and then call the number of unique values remaining to give us our vocabulary size:

```
In [26]: #converts the vocab arrays to dataframes
         yelp = pd.DataFrame(yelp_vocab)

         amzn = pd.DataFrame(amzn_vocab)

         imdb = pd.DataFrame(imdb_vocab)
```

```
In [41]: #places all vocab lists into a single data frame called 'vocab'
         vocab = pd.concat([amzn, imdb, yelp])

In [43]: #checks for sum of duplicated words in vocab
         vocab.duplicated().sum()

Out[43]: 1474

In [46]: #removes duplicates and gives us the number of unique words / vocab size
         vocab.drop_duplicates(inplace=True)
         vocab.nunique()

Out[46]: 0    4541
         dtype: int64
```

As we can see above, the total vocabulary size for our dataset is 4,541 words.

As we move forward with our Keras model, we will need to both tokenize our vocabulary list as well as select word embedding length and maximum embedding sequence length. We will discuss this below, but we will likely use a word embedding length of 4,542 (which is 0-4,541), which is our vocabulary size + 0. For our maximum sequence length, we will likely set this to 7,944, due to seeing above, that our maximum character length in the 'sentence' column was 7,944 characters.

B2.

As we will be utilizing machine learning to analyze the sentiment of our words, we cannot just feed the neural network strings and characters. It is required that we 'tokenize' the words in our vocabulary list in order to assign each word a unique integer (Brownlee, 2021). This step is critical before we can move forward with word embedding.

To complete the tokenization process, we will utilize the Tokenizer API from Keras.preprocessing. We will define the tokenizer with 'num_words', which will be our vocabulary size of 4,541. We will then fit the tokenizer on our training and testing data (which we will cover below in section B5), by converting the text through the tokenizer, and then define the vocab_size, along with adding 1 for the reserved 0 index. We can then print a row from our training data, along with the tokenized values.

```
In [92]: from keras.preprocessing.text import Tokenizer

         tokenizer = Tokenizer(num_words=4541)
         tokenizer.fit_on_texts(sentences_train)

         X_train = tokenizer.texts_to_sequences(sentences_train)
         X_test = tokenizer.texts_to_sequences(sentences_test)

         vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0 index

         print(sentences_train[5])
         print(X_train[5])

         It failed to convey the broad sweep of landscapes that were a great part of the original.
         [9, 910, 8, 911, 1, 912, 913, 5, 914, 14, 48, 2, 50, 178, 5, 1, 369]
```

Here, we can see the fifth row from our 'sentences_train' data, along with the tokenized values for that sentence. We can see that the more common the word in the text, the lower the value. For instance, the word "the" is given a value of 1, while the word "failed" is given a value of 910.

B3.

Due to the fact that most of the reviews in the 'sentence' column of our data frame are of varying length, we will need to utilize Keras' padding function in order to make all of the sequences the same length. This is accomplished by adding zeros to pad the sequence lengths so that they will all be the same length.

The padding function adds the zeros to the beginning of the text sequence by default, but we will instead append these, by specifying "padding='post'.

We know that our longest value in the 'sentence' column is 7,944 characters. We will use this number as our maximum length for our padding function, and apply this to our training and testing set:

```
In [112]: maxlen = 7944

          X_train = pad_sequences(X_train, padding = 'post', maxlen = maxlen)
          X_test = pad_sequences(X_test, padding = 'post', maxlen = maxlen)
```

As we can see, using the same selected row as above, the padding function has added several zeros to the end of the sequence. It is abbreviated, due to being 7,944 characters long, however we verify that the zeros have been added to the end of the sequence:

```
In [109]: X_train[5]

Out[109]: array([  9, 910,   8, ...,   0,   0,   0])
```

B4.

For this analysis we will try to determine two categories of sentiment using our dataset; Positive or Negative sentiment.

For an activation function for the final dense layer of our neural network, we will be utilizing a sigmoid activation function for our final dense layer of our neural network. Sigmoid is a nonlinear function, so the output of this will be the weighted sum of inputs (Saeed, 2021).

B5.

In addition to the steps mentioned above in section B1, we utilize Sci-Kit Learn's train/test split in order to split our data into training and testing sets. For our training and testing sets, we will use the concatenated data frame of all the reviews, using only the 'sentence' columns where 'source' = 'yelp' as our testing. Our y value for training and testing will be the 'label' column where the 'source' = 'yelp'. Our training and testing

split sizes will be 25% for testing, and 75% for training, and validation data being the combination of our X_test and y_test.

```
In [25]: from sklearn.model_selection import train_test_split

         df_yelp = df[df['source'] == 'yelp']

         sentences = df_yelp['sentence'].values
         y = df_yelp['label'].values

         sentences_train, sentences_test, y_train, y_test = train_test_split(sentences, y, test_size=0.25, random_state=1000)
```

```
In [26]: from sklearn.feature_extraction.text import CountVectorizer

         vectorizer = CountVectorizer()
         vectorizer.fit(sentences_train)

         X_train = vectorizer.transform(sentences_train)
         X_test  = vectorizer.transform(sentences_test)
         X_train

Out[26]: <747x1708 sparse matrix of type '<class 'numpy.int64'>'
                 with 7327 stored elements in Compressed Sparse Row format>
```

B6.
Please see the attached 'D213_Task2.csv' file.

Part III: Network Architecture

C1.
We can see our model summary below:

```
In [46]: from keras.models import Sequential
         from keras import layers

         embedding_dim = 50

         model = Sequential()
         model.add(layers.Embedding(input_dim=vocab_size,
                                    output_dim=embedding_dim,
                                    input_length=maxlen))
         model.add(layers.GlobalMaxPool1D())
         model.add(layers.Dense(10, activation='relu'))
         model.add(layers.Dense(1, activation='sigmoid'))
         model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
         model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 7944, 50)          123200
_________________________________________________________________
global_max_pooling1d (Global (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 10)                510
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 123,721
Trainable params: 123,721
Non-trainable params: 0
_________________________________________________________________
```

C2.
As we can see in our model summary, we utilized four layers:

- An embedding layer utilizing our vocab_size, maxlen and embedding_dim that we had defined earlier
- A GlobalMaxPool1d layer, which takes the maximum value of all features in the pool of each feature dimension (Real Python, n.d.)
- 1 Dense layer with the activation function of RELU (Rectified Linear Unit)
- 1 Dense layer with the activation function of Sigmoid

We can also note our total number of parameters: 123,721, which is our vocab_size multiplied by the 'embedded_dim' value.

C3.
Let us discuss the decisions in choosing our model's hyperparameters above.

Activation Functions
We have selected two activation functions within our model, the first being Rectified Linear Unit, or RELU. RELU will pass the input through if the value is positive or a zero if the value is less than, or equal to, zero. RELU is widely regarded as the default activation function for most neural networks, so its inclusion is due to how RELU can boost accuracy of any neural network model (Brownlee, 2020).

We have also chosen to include Sigmoid as our final dense layer activation function. Sigmoid is a popular logistic function that is excellent when being used in a model that predicts probability. The function takes the inputs into the function and outputs a value between 0 and 1. Sigmoid being a nonlinear function is preferred for neural networks where the model needs to learn more complex data structures, such as sentiment analysis (Brownlee, 2020).

Number of Nodes per Layer
When deciding the number of nodes per layer, as well as number of layers, there does not seem to be any generally agreed upon answer. The number of layers and nodes tends to be determined by complexity of the problem and, simply, random experimentation (Brownlee, 2019). We have decided on four layers with 10 nodes in our first dense layer and 1 node in our second dense layer. This can be seen as a starting point that, depending on model performance, can be altered and refined, based on our outcomes.

Optimizer
We have selected 'Adam' as our optimizer for this model due to the fact that 'Adam' is currently the most common optimizer used for neural networks, and has very good performance across assorted problems (Real Python, n.d.).

Stopping Criteria
For our model, we defined a number of epochs for the model to iterate through before stopping. We set this value at 50 initially, but lowered this value to 10 after seeing that beyond 10 epochs, the loss of the validation data began to rise after decreasing prior to ~10 epochs, suggesting the model was overfitting (Real Python, n.d.).

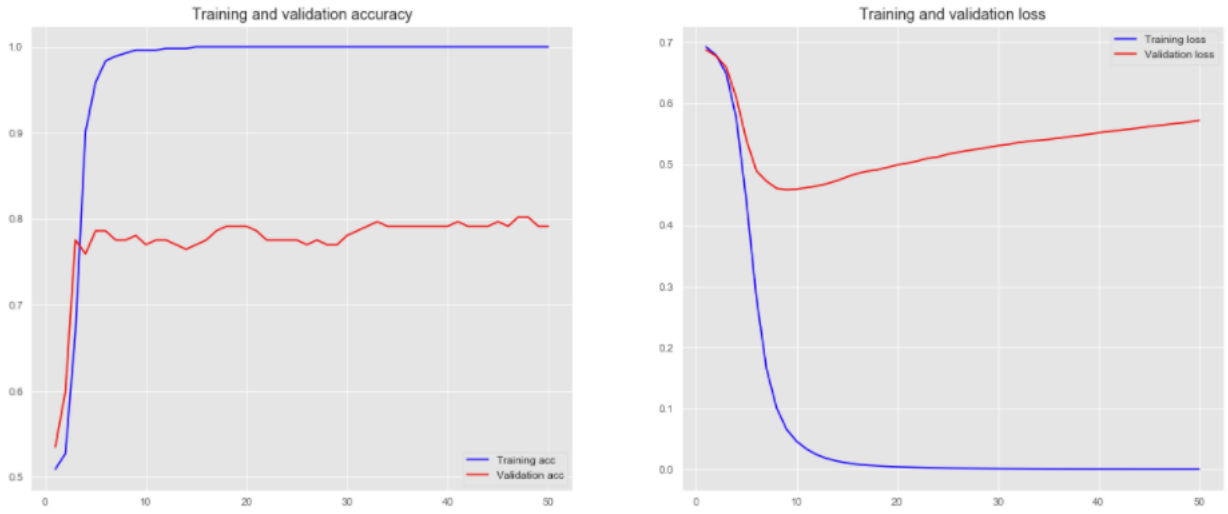
Evaluation Metric

To evaluate the model, we have chosen to look at accuracy, which Keras defines as how often the predictions equal the labels (Keras Documentation: Accuracy Metrics, n.d.). In the case of our model, it will be how often the predicted labels for positive/negative sentiment reviews match the validation labels for positive/negative sentiment reviews.

Part IV: Model Evaluation

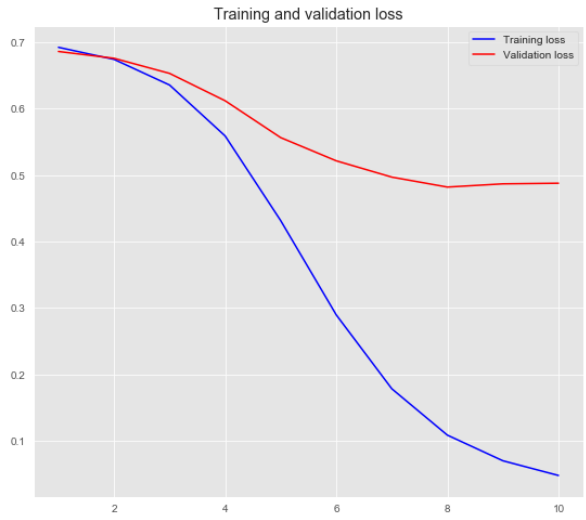
D1 - D4.

For our model, as briefly discussed above, we opted to stop the training of the model after 10 epochs. We decided this because we noticed that our model was exhibiting signs of overfitting. This can be seen here:



On the right-hand image, we can see that the validation data loss decreases during the first ~10 epochs and then begins to rise again and continues increasing through the remaining 40 epochs. This suggests the model is overfitting.

After setting the number of epochs to 10, we can see the model stops training before this increased loss in validation data occurs:



A challenge of training neural network models is to train the model enough to correctly identify the variables it is training and predicting on, but not overtrain the model to the point where it overfits the training data, leading to an increase in generalization errors (Brownlee, 2019a).

There are several ways to utilize stopping criteria instead of defining a number of epochs, such as defining a threshold for generalization error. While training through a large number of epochs, the model will stop training if the performance of the model on the validation set begins to degrade beyond the specified threshold (Brownlee, 2019a).

While determining the model's number of epochs by running the model several times and determining when the overfitting begins to occur is computationally wasteful, and not the most efficient way to determine the stopping criteria and prevent the model from overfitting, in the case of this model where it is being trained on a relatively small dataset, it was a relatively fast and accurate way to achieve the same results that a more complex stopping criteria would give.

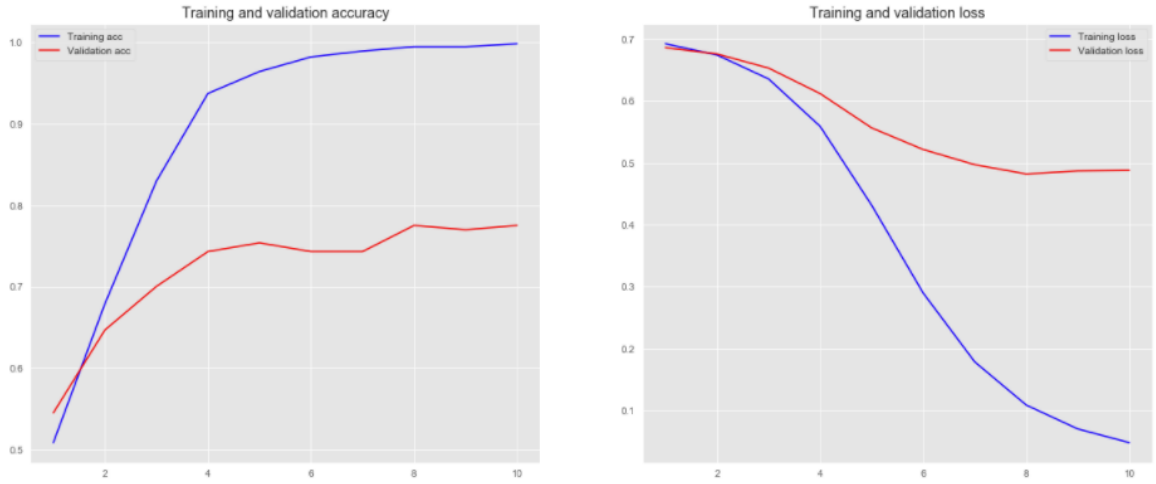
On a larger and more complex data set with more layers and nodes, we would definitely want to look into defining stopping criteria more concretely because the time required to run our model could be exponentially longer.

Again, here we can see the final run of our model with 10 epochs:

```
In [47]: history = model.fit(X_train, y_train, epochs=10, verbose=False, validation_data=(X_test, y_test), batch_size=10)

         loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
         print("Training Accuracy: {:.4f}".format(accuracy))
         loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
         print("Testing Accuracy:  {:.4f}".format(accuracy))
         plot_history(history)

         Training Accuracy: 0.9982
         Testing Accuracy:  0.7754
```



We can see here that our model ended with a training accuracy of 77.5%, meaning it correctly predicted the positive or negative sentiment of the reviews 77.5% of the time and that our model would be able to accurately predict the sentiment of new review data with an accuracy of 77.5%. This accuracy is not bad for a quickly built neural network that has not been refined, other than to ensure we are not overfitting the model.

Part V: Summary and Recommendations

E.

We save our Keras neural network model with the following code:

```
In [73]: #saves neural net model
         model.save("model_backup.h5")
         print("Saved model")

         Saved model to disk
```

F.

Our relatively simple, sequential neural network seemed to work rather well for this analysis. The accuracy of our model was good, but not great, and clearly has room to improve. Of course, it should be noted that our dataset was relatively small, utilizing only a small number of hidden layers and dense layers. On a much larger dataset that is trying to predict on a larger testing sample, we would likely need to use different types of neural networks, and a much larger number of layers depending on the question we would be trying to answer.

G.

Based on our model's accuracy, we have demonstrated that it is possible to predict customer sentiment from reviews by utilizing NLP techniques with a neural network machine learning model. We saw that our model had 77.5% accuracy, which is good, but not great, and has clear room for improvement.

We would recommend to stakeholders that the model be further developed, by experimenting with different types of neural network models, along with expanding the number of nodes and layers to determine if these alterations to the model will improve the model's performance.

Third-party code referenced from:

Real Python. (n.d.). *Practical Text Classification With Python and Keras*. Retrieved November 3, 2021, from https://realpython.com/python-keras-text-classification/

References

Brownlee, J. (2019a, August 6). *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks*. Machine Learning Mastery. Retrieved November 3, 2021, from https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models

Brownlee, J. (2019b, August 6). *How to Configure the Number of Layers and Nodes in a Neural Network*. Machine Learning Mastery. Retrieved November 3, 2021, from https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

Brownlee, J. (2020, August 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Machine Learning Mastery. Retrieved November 3, 2021, from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, J. (2021, February 1). *How to Use Word Embedding Layers for Deep Learning with Keras*. Machine Learning Mastery. Retrieved November 3, 2021, from https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Keras documentation: Accuracy metrics. (n.d.). Keras. Retrieved November 3, 2021, from https://keras.io/api/metrics/accuracy_metrics/

Real Python. (n.d.). *Practical Text Classification With Python and Keras*. Retrieved November 3, 2021, from https://realpython.com/python-keras-text-classification/

Saeed, M. (2021, August 18). *A Gentle Introduction To Sigmoid Function*. Machine Learning Mastery. Retrieved November 3, 2021, from https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/

Vocabulary size in CSV file. (2013, January 21). Stack Overflow. Retrieved November 3, 2021, from https://stackoverflow.com/questions/14441846/vocabulary-size-in-csv-file