ACADEMIC SESSION 2021/2022, SEMESTER 1

SCHOOL OF COMPUTER SCIENCES

CPC353: NATURAL LANGUAGE PROCESSING

(Assignment 2)

Name : Tan Chi Feng

Matric Number : 147515

Lecturer : Dr. Tan Tien Ping

Submission Date : 31 JANUARY 2022

Video Link : https://youtu.be/VdnUnVrA-fA

Project Report

To begin with, we first import all the libraries and packages need.

```
In [1]: import os
        import sys
        import numpy as np
        import re
        import string

        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, LSTM, Flatten, Bidirect
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences
        from tensorflow.keras.optimizers import Adam, SGD
        from tensorflow.keras.utils import to_categorical
        import matplotlib.pyplot as plt
```

Data Preparation

In this phase, our objective is to prepare the training, validation, and test datasets so that they can fit into the Bidirectional LSTM layers. LSTM layers require data of 3 dimensions, since we could not use embedding layers to output this 3-dimensions data, I will use python to make it happen, details will be discussed in the section below.

1. Reading the train.csv, validation.csv and test.csv dataset into data frame with pandas packages.

```
In [2]: import pandas as pd

        training_data = pd.read_csv("/Volumes/GoogleDrive/My Drive/School of Comput
        val_data = pd.read_csv("/Volumes/GoogleDrive/My Drive/School of Computer Sc
        test_data = pd.read_csv("/Volumes/GoogleDrive/My Drive/School of Computer S
```

2. Renaming the columns of training, validation and test data frame to appropriate column name.

```
In [3]: # Rename the columns
        training_data.columns = ['Headlines', 'Sentiment']
        val_data.columns = ['Headlines', 'Sentiment']
        test_data.columns = ['Headlines', 'Sentiment']
```

3. Performing data cleaning and preprocessing by removing stop words in all datasets. HTML tags like "<" and ">" will also be removed. Method of performing stop words

removing on Python is inspired by Ketan Vaidya on his articles on Toward Data Science. [1]

In [4]:
```
stopwords = [ "a", "about", "above", "after", "again", "against", "all", "am", "an", "and", "any", "are", "as", "at", "
              "been", "before", "being", "below", "between", "both", "but", "by", "could", "did", "do", "does", "doing",
              "each", "few", "for", "from", "further", "had", "has", "have", "having", "he", "he'd", "he'll", "he's", "h
              "here's", "hers", "herself", "him", "himself", "his", "how", "how's", "i", "i'd", "i'll", "i'm", "i've", "
              "is", "it", "it's", "its", "itself", "let's", "me", "more", "most", "my", "myself", "nor", "of", "on", "on
              "other", "ought", "our", "ours", "ourselves", "out", "over", "own", "same", "she", "she'd", "she'll", "she
              "so", "some", "such", "than", "that", "that's", "the", "their", "theirs", "them", "themselves", "then", "t
              "these", "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to", "too", "unde
              "very", "was", "we", "we'd", "we'll", "we're", "we've", "were", "what", "what's", "when", "when's", "where
              "which", "while", "who", "who's", "whom", "why", "why's", "with", "would", "you", "you'd", "you'll", "you'
              "your", "yours", "yourself", "yourselves" ]
```

In [51]:
```
def remv_stopwords(data):
  data['Headlines'] = data['Headlines'].apply(lambda x : ' '.join([word for word in x.split() if word not in (stopwords
  return data

def remv_htmltags(string):
    result = re.sub('<.*?>','',string)
    return result

training_data = remv_stopwords(training_data)
training_data['Headlines']= training_data['Headlines'].apply(lambda cw : remv_htmltags(cw))
training_data['Headlines'] = training_data['Headlines'].str.replace('[{}]'.format(string.punctuation), ' ')

val_data = remv_stopwords(val_data)
val_data['Headlines']= val_data['Headlines'].apply(lambda cw : remv_htmltags(cw))
val_data['Headlines'] = val_data['Headlines'].str.replace('[{}]'.format(string.punctuation), ' ')

test_data = remv_stopwords(test_data)
test_data['Headlines']= test_data['Headlines'].apply(lambda cw : remv_htmltags(cw))
test_data['Headlines'] = test_data['Headlines'].str.replace('[{}]'.format(string.punctuation), ' ')
```

Building Dictionary Using Pretrained Word Vectors from GloVe

4. Building a dictionary named "embeddings_index" by using the pretrained word vectors obtained from https://nlp.stanford.edu/projects/glove/ (specifically the 'glove.6B.300d.txt').

   Since it is a dictionary, for instance, if we call embeddings_index['the'], since we supply the key 'the' to the dictionary, the dictionary will return the corresponding word vector that associate with the word 'the'.

   From the output we can see that there is a total of 400000 word vectors found in the pretrained word vectors.

In [6]:
```
with tf.device('/cpu:0'):
    working_dir = "/Volumes/GoogleDrive/My Drive/School of Computer Science/Year 3/Semester 1/4. CPC353 Natural Languag

    embeddings_index = {}

    with open(os.path.join(working_dir, 'glove.6B.300d.txt')) as file:
        for line in file:
            word, coefs = line.split(maxsplit=1)
            coefs = np.fromstring(coefs, 'f', sep=' ')
            embeddings_index[word] = coefs

    print('There are a total of %s word vectors found in glove.6B.300d.txt.' % len(embeddings_index))
```

```
Metal device set to: Apple M1 Pro

systemMemory: 16.00 GB
maxCacheSize: 5.33 GB

There are a total of 400000 word vectors found in glove.6B.300d.txt.
```

5. For instance, we can test the dictionary that we build by providing a key, in this example we provide the key “dr”, an array of 300 word vectors corresponding to the ‘dr’ is returned.

```
In [7]: embeddings_index['dr']

Out[7]: array([-0.074607 , -0.11633  ,  0.36245  ,  0.36715  , -0.76794  ,
               -0.30341  , -0.18367  ,  0.13902  ,  0.36768  , -1.1403   ,
                0.15194  , -0.58007  , -0.11208  , -0.076548 ,  0.090214 ,
                0.43275  ,  0.024418 ,  0.20571  ,  0.23143  , -0.62702  ,
               -0.32204  ,  0.17888  ,  0.18504  , -0.014398 ,  0.015564 ,
                0.5972   ,  0.070547 , -0.049865 ,  0.13624  , -0.62059  ,
               -0.27566  ,  0.12168  ,  0.075928 ,  0.12725  , -0.19275  ,
                0.23169  ,  0.87206  ,  0.55183  , -0.39115  , -0.54558  ,
               -0.03112  , -0.60598  , -0.046408 ,  0.11338  ,  0.046661 ,
               -0.1819   , -0.28307  , -0.029926 , -0.46608  , -0.39512  ,
               -0.16182  ,  0.38471  , -0.16378  ,  0.25356  ,  0.10304  ,
                1.0363   ,  0.2292   ,  0.43539  ,  0.17291  ,  0.060108 ,
               -0.19593  ,  0.32616  , -0.02618  ,  0.59961  ,  0.42063  ,
               -0.17343  , -0.46433  , -0.010907 , -0.013383 , -0.3362   ,
               -0.1653   , -0.1498   ,  0.16513  ,  0.38764  ,  0.25905  ,
               -0.33554  ,  0.66784  ,  0.056679 , -0.21026  ,  0.014089 ,
                0.68027  ,  0.11919  ,  0.10883  ,  0.44258  ,  0.29052  ,
               -0.16496  ,  0.14129  , -0.029832 ,  0.30671  , -0.54124  ,
               -0.27288  ,  0.057294 ,  0.0053464,  0.088504 ,  0.58858  ,
```

Tokenizing Text Samples in Training Dataset

6. We tokenize every row of headlines in the dataset, and we assume that each row of headlines has a maximum of 100 words.

   Each unique word will be assigned a unique token, for instance, all the word "increase" will have a token "678".

   For headlines that have less 100 words, the remaining will be assigning a token "0".

   From the output we can see that there is a total of 24392 tokens found in the training dataset.

```
In [8]: MAX_NUM_WORDS = 28000

        # Vectorizing the text samples into a 2D integer tensor
        tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
        tokenizer.fit_on_texts(training_data.Headlines[0:])
        sequences = tokenizer.texts_to_sequences(training_data.Headlines[0:])

        word_index = tokenizer.word_index
        print('There are %s unique tokens in the training dataset.' % len(word_index))

        MAX_SEQUENCE_LENGTH = 100

        data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH, dtype=object)

        labels = to_categorical(training_data.Sentiment - training_data.Sentiment.min())
        print('Shape of data tensor:', data.shape)
        print('Shape of label tensor:', labels.shape)

There are 24392 unique tokens in the training dataset.
Shape of data tensor: (61692, 100)
Shape of label tensor: (61692, 3)
```

7. For instance, the array "data" consist of all the tokenized words, data[0] which corresponds to the first row of headlines in the training data, will return 100 tokens, since we assume that all headlines will have a maximum of 100 words, empty space will be given token "0". When we compare it to the first headline training data, we can see that the word "dr" has a token of "384".

```
In [9]: data[0]

Out[9]: array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 384, 102, 230, 1206, 233, 4601, 2973, 2257], dtype=object)

In [10]: training_data

Out[10]:
```

	Headlines	Sentiment
0	Dr M now interim PM Agong accepts resignation	0
1	MMAG s M Jets targets annual cargo volume grow...	1
2	Civil servants will continue provide quality ...	0

Building Embedding Matrix for Training Dataset

8. Building an embedding matrix for each token in the training dataset by mapping each token to the corresponding word vectors in the "embedding_index" dictionary that we build.

   For instance, for the word 'dr', its token number is 384, by looking up to the dictionary, embeddings_index['dr'], a corresponding word vector of 300d will be returned and mapped to token '384'.

```
In [11]: EMBEDDING_DIM = 300

         print('Preparing embedding matrix.')

         # prepare embedding matrix
         num_words_train = min(MAX_NUM_WORDS, len(word_index) + 1)
         embedding_matrix = np.zeros((num_words_train, EMBEDDING_DIM))
         for word, i in word_index.items():
             if i >= MAX_NUM_WORDS:
                 continue
             embedding_vector = embeddings_index.get(word)
             if embedding_vector is not None:
                 # words not found in embedding index will be all-zeros.
                 embedding_matrix[i] = embedding_vector
         print(embedding_matrix.shape)

         Preparing embedding matrix.
         (24393, 300)
```

Generating 3 dimensional inputs for LSTM layers

9. The two cells below are used to generate 3 dimensional inputs for the LSTM layer of our machine learning model. Since we could not use embedding layer to generate this 3-dimensional inputs, I figure out a way to achieve this using Python.

   The 2 for loops in the first cell are to loop through every single token in each row of headlines. For each token, we will look up to the dictionary (embeddings_index) and find its corresponding word vector. Then, we will replace this token with the word vector returned. Note that this operation is viable because I initially set the data type of each token to be an object, instead of float or int, that's why we can assign an array to replace each token directly (this is not possible if the token is of data type int or float).

   As a result, the input of our training set, x_train, will have a dimension of (61692, 100, 300), which represents 61692 rows of headlines in the training set, for each row of headline there are 100 tokens, for every token a word vector of 300d is assigned.

```
In [12]: for i in range(61692):
             for j in range(100):
                 data[i][j] = embedding_matrix[int(data[i][j])]

In [13]: x_train = np.zeros((61692,100,300))
         for j in range(61692):
                 x_train[j] = np.stack(data[:][j])

In [14]: x_train.shape
Out[14]: (61692, 100, 300)
```

For Validation and Test Dataset

10. Step 6 to 9 will be repeated for validation and test dataset, since it is quite straight-forward, therefore the process will not be discussed in this report.

Finalized Data Preparation

11. Assigned respective labels to y_train, y_val and y_test. Final check if all the shape or dimension is correct before using them for training the Bidirectional LSTM classifier.

```
In [23]: x_train.shape
         x_val.shape
         x_test.shape
Out[23]: (1000, 100, 300)

In [24]: y_train = labels
         y_val = valLabels
         y_test = testLabels

In [25]: y_train.shape
Out[25]: (61692, 3)
```

Model Building

12. Create a sequential model to prepare a plain stack of layers where each layer has exactly one input tensor and one output tensor.

```
In [26]: model = Sequential()
```

13. Adding Bidirectional LSTM with 128 memory units as the first layer [2], following by a flatten layer (this is required when the return_sequences are set to true, where the lstm model will return the hidden state output for each input time step), a Dense Layer with 128 neurons and last but not least a Dense Layer with 'softmax' activation function applied and it is set to output 3 types of label (since our sentiment label consist of 'negative', 'neutral' and 'positive'.

    Other than the final Dense Layer, all other LSTM and Dense layers are using 'relu' activation function.

    Dropout layers with rate of 0.2 are applied after the first Bidiretional LSTM layer and the second Dense layer to prevent the model from overfitting the training data.

```
In [27]: model.add(Bidirectional(LSTM(128, input_shape=(x_train.shape[1:]), activation='relu', return_sequences=True)))
         model.add(Dropout(0.2))
         model.add(Flatten())
         model.add(Dense(128, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(len(y_train[0]), activation='softmax'))
```

14. Compiling the model with Adam optimzizer and loss functiom of categorical_crossentropy.

```
In [28]: opt1 = SGD(learning_rate=0.01, momentum=0.9)
         opt2 = Adam(learning_rate=1e-3, decay=1e-5)
```

```
In [29]: # compile the model
         model.compile(loss='categorical_crossentropy',
                       optimizer=opt2,
                       metrics=['accuracy'])
```

15. Fitting the training data into the model to train the model with batch size of 1024 and 5 epochs.

```
In [30]: with tf.device('/cpu:0'):
             training_model = model.fit(x_train, y_train,
                                        batch_size=1024,
                                        epochs=5,
                                        validation_data = (x_val, y_val))
```

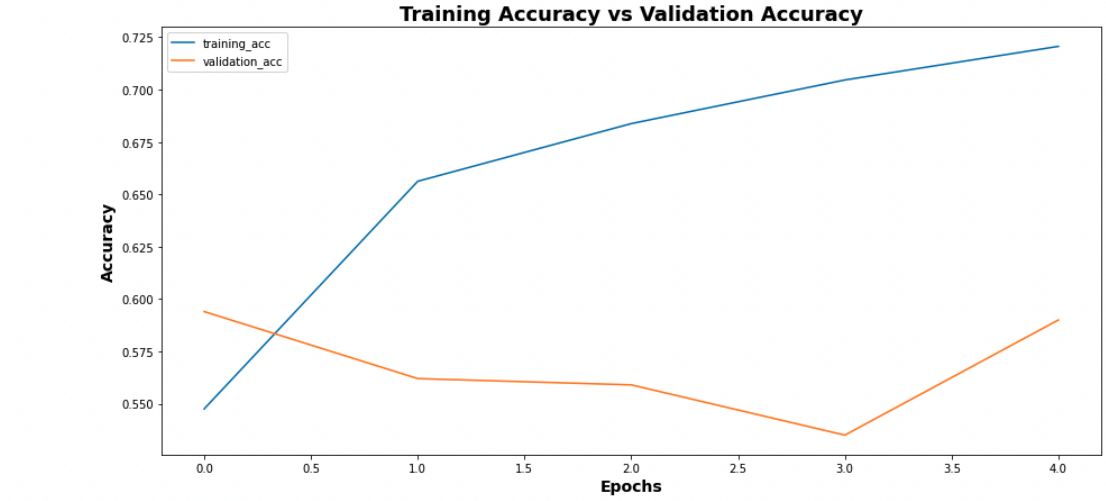
```
2022-01-29 19:10:02.107905: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0
Hz

Epoch 1/5

2022-01-29 19:10:02.622568: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] Plugin opti
mizer for device_type GPU is enabled.

61/61 [==============================] - ETA: 0s - loss: 0.9323 - accuracy: 0.5475

2022-01-29 19:10:50.181109: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:112] Plugin opti
mizer for device_type GPU is enabled.

61/61 [==============================] - 48s 776ms/step - loss: 0.9323 - accuracy: 0.5475 - val_loss: 0.8261 - val_ac
curacy: 0.5940
Epoch 2/5
61/61 [==============================] - 48s 784ms/step - loss: 372.9150 - accuracy: 0.6562 - val_loss: 0.8799 - val_
accuracy: 0.5620
Epoch 3/5
61/61 [==============================] - 50s 818ms/step - loss: 0.7359 - accuracy: 0.6838 - val_loss: 0.9085 - val_ac
curacy: 0.5590
Epoch 4/5
61/61 [==============================] - 48s 788ms/step - loss: 0.6969 - accuracy: 0.7046 - val_loss: 0.9570 - val_ac
curacy: 0.5350
Epoch 5/5
61/61 [==============================] - 48s 788ms/step - loss: 0.6651 - accuracy: 0.7206 - val_loss: 0.8663 - val_ac
curacy: 0.5900
```

Model Evaluation

16. Plotting a line graph to see how training and validation accuracy increase after each epoch. From the output we can see that the training accuracy increases after each epoch while the validation accuracy decreases in the first 4 epochs but increases in the fifth epcohs. This implies that 5 epochs are great for training the model, in fact after few trials with different number of epochs, 5 epochs give the optimum training and test accuracy, more epochs might cause the model to overfit and consume unnecessary computation time and power.

```
In [48]: #plot accuracy
         plt.figure(figsize=(15, 7))
         plt.plot(range(5), training_model.history['accuracy'])
         plt.plot(range(5), training_model.history['val_accuracy'])
         plt.legend(['training_acc', 'validation_acc'])
         plt.xlabel("Epochs", fontweight="bold", fontsize="14")
         plt.ylabel("Accuracy", fontweight="bold", fontsize="14")
         plt.title('Training Accuracy vs Validation Accuracy', fontweight="bold", fontsize="18")
```

```
Out[48]: Text(0.5, 1.0, 'Training Accuracy vs Validation Accuracy')
```



17. Evaluate the model with test dataset and display the corresponding accuracy and loss.

```
In [49]: with tf.device('/cpu:0'):
             loss, accuracy = model.evaluate(x_test, y_test, verbose=0)

             print('Accuracy (Test Set): %f' % (accuracy))
             print('Loss     (Test Set): %f' % (loss))

Accuracy (Test Set): 0.615000
Loss     (Test Set): 0.845002
```

18. The model summary is displayed.

```
In [34]: model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 100, 256)         439296
 l)

 dropout (Dropout)           (None, 100, 256)          0

 flatten (Flatten)           (None, 25600)             0

 dense (Dense)               (None, 128)               3276928

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 3,716,611
Trainable params: 3,716,611
Non-trainable params: 0
_________________________________________________________________
```

Model Testing

19. We can try to predict the output based on any given headlines in the test sets.

```
In [35]: with tf.device('/cpu:0'):
             output = model.predict(x_test)
```

20. From the output, let's take the first headline as an example, the array of index 2 get the highest probability, which is 0.6575, this tells us that the first headline is most probably a positive news headline. (Index 0 – "Negative", Index 1 – "Neutral", Index 2 – "Positive")

```
In [36]: output

Out[36]: array([[0.13466485, 0.20781979, 0.65751535],
                [0.009301  , 0.7471878 , 0.24351121],
                [0.02135628, 0.8545073 , 0.12413646],
                ...,
                [0.5479214 , 0.0211898 , 0.43088883],
                [0.6611812 , 0.26026696, 0.07855193],
                [0.01668142, 0.09117854, 0.89214003]], dtype=float32)
```

21. We can use the argmax() function to print the index with the highest probability for each headline.

```
In [37]: result = np.argmax(output, axis=1)
         print(result)
```

```
[2 1 1 0 1 1 2 2 2 0 1 1 1 2 0 0 2 0 0 1 1 0 0 2 2 2 0 1 1 0 1 1 2 1 0 1 2
 2 1 1 1 1 0 0 0 0 1 1 0 2 1 1 2 1 0 1 0 1 1 2 2 0 0 0 2 1 1 2 2 1 1 2 1 1
 2 2 1 1 2 1 1 2 2 1 2 1 2 2 1 1 1 2 2 2 2 2 1 2 0 1 1 2 1 2 2 1 2 1 1 2 0
 1 1 0 0 2 2 1 0 2 1 1 2 1 0 0 2 2 1 2 0 2 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 2
 1 0 0 2 2 2 2 1 2 2 1 2 1 1 0 2 0 2 1 1 1 2 2 1 1 0 2 2 1 2 1 1 2 1 2 2 2
 2 2 2 2 2 2 2 2 0 2 1 1 2 2 0 2 0 2 2 2 2 1 1 2 2 0 2 0 2 2 0 1 1 0 1 2 2
 1 2 2 2 1 1 1 2 0 0 2 0 1 0 1 1 2 0 2 2 2 1 2 0 1 2 0 1 1 1 2 1 0 2 2 1 1
 2 1 0 2 1 0 1 2 1 2 1 0 1 2 2 2 1 2 1 1 0 0 1 2 0 1 1 2 2 1 0 1 1 2 2 2 0
 2 2 0 1 1 2 2 2 2 2 2 1 0 1 1 1 2 1 1 2 2 2 1 0 1 2 1 2 2 0 2 1 1 1 0 2 1
 1 0 0 2 0 1 1 1 1 2 2 0 2 0 2 0 2 2 1 0 2 0 2 1 1 1 1 2 2 1 1 1 1 0 1 0 2
 2 0 0 2 0 0 2 0 1 2 1 1 1 2 1 2 2 1 1 0 2 2 0 2 0 2 1 1 2 0 2 1 1 1 2 1 2 2 0
 1 2 2 0 1 1 2 2 1 2 0 2 1 0 0 0 0 1 1 2 2 2 2 0 2 1 1 0 1 2 1 0 2 0 1 1 2 2
 0 2 2 0 2 2 1 2 0 2 1 1 2 1 2 2 0 1 2 0 2 1 1 2 0 1 0 0 2 2 2 2 2 2 1 2 1
 0 2 1 2 0 1 2 1 0 1 1 2 1 2 2 2 2 1 1 2 0 2 2 2 2 2 2 0 1 0 1 0 1 0 1 0 2 1
 1 1 1 0 1 0 1 1 0 2 1 2 1 1 2 2 1 2 0 2 0 1 1 2 2 2 2 2 2 0 2 2 1 2 2 1 0 1 2
 0 0 0 0 2 2 2 0 0 2 2 2 2 2 0 2 1 2 1 2 0 2 0 2 2 1 2 0 0 0 0 0 0 0 1 1 1 1
 2 0 2 0 0 2 0 2 0 2 2 1 0 2 1 2 2 2 2 2 1 2 0 2 0 0 2 2 2 1 2 2 1 2 1 0 1
 1 2 1 1 1 0 0 2 2 0 2 2 2 0 2 0 0 2 1 2 2 1 0 2 2 0 2 0 2 0 2 2 1 1 2 2 2 1 2
 0 2 2 2 1 2 0 1 2 0 1 0 0 2 2 0 0 2 1 2 2 2 2 2 2 2 2 1 2 1 0 0 1 0 0 0 1
 2 2 0 2 2 0 2 0 2 0 0 1 2 1 0 2 1 2 2 2 1 2 0 1 0 2 2 0 1 2 2 2 1 1 2 2 1
 2 0 2 1 0 2 2 1 0 1 1 0 1 1 2 1 0 1 0 1 1 2 2 1 0 0 2 1 2 2 2 2 2 1 2 1 2
 1 0 2 0 2 2 1 1 1 2 2 2 1 2 2 0 1 1 1 0 2 0 1 1 2 2 2 2 2 2 2 2 0 2 0 2 2
 2 2 2 2 2 2 1 0 2 2 2 1 2 2 1 2 2 2 0 1 0 0 2 2 2 2 1 1 0 0 0 2 2 2 2 1 1
 1 1 0 2 2 2 2 2 2 1 2 0 2 2 0 0 0 0 1 0 2 1 0 2 0 2 2 0 1 2 1 0 2 2 0 0 1
 1 2 1 2 2 2 2 1 2 0 1 2 0 2 2 2 0 2 2 2 1 0 2 0 0 1 0 2 0 2 2 2 2 1 2 0 0
 1 1 2 1 2 2 0 2 2 2 2 2 2 2 0 0 2 2 0 2 2 2 2 2 1 2 0 2 1 0 2 0 2 2 0 2 2
 1 0 0 1 2 2 2 0 2 2 1 1 0 0 0 2 2 0 1 2 1 2 0 1 0 0 2 0 0 2 2 0 2 2 1 0 0
 2]
```

22. Finally, we create an array named sentiment_test and assign “negative”, “neutral” and “positive” to array of index 0, 1, 2 respectively.

```
In [38]: sentiment_test = []
         for i in range(len(result)):
             if result[i] == 0:
                 sentiment_test.append("negative")
             elif result[i] == 1:
                 sentiment_test.append("neutral")
             else:
                 sentiment_test.append("positive")
```

23. When we print the array “sentiment_test”, we can see the sentiment result (negative, neutral or positive) for each row of headline in the test set.

```
In [50]: sentiment_test
Out[50]: ['positive',
          'neutral',
          'neutral',
          'negative',
          'neutral',
          'neutral',
          'positive',
          'positive',
          'positive',
          'negative',
          'neutral',
          'neutral',
          'neutral',
          'positive',
          'negative',
          'negative',
```

Conclusion

This Bidirectional LSTM classifier is capable of performing sentiment analysis with an acceptable accuracy based on the dataset that we tested on. We can further improve the accuracy by training the model with more accurate training sets by having experts to annotate the headlines. Ultimately, the quality of the training sets plays a very important role in determining the accuracy of the model.

References

[1] "Sentiment Analysis using LSTM and GloVe Embeddings | by Ketan Vaidya | Towards Data Science." https://towardsdatascience.com/sentiment-analysis-using-lstm-and-glove-embeddings-99223a87fe8e (accessed Jan. 30, 2022).

[2] "How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras." https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/ (accessed Jan. 30, 2022).