

**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.

Graphical user interface, text, application

Description automatically generated

**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.  
   Text

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

A picture containing text

Description automatically generated

1. 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]

Text

Description automatically generated

**Building Dictionary Using Pretrained Word Vectors from GloVe**

1. 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.

Text

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

1. 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.

**Table

Description automatically generated**

**Tokenizing Text Samples in Training Dataset**

1. 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.

Graphical user interface, text, application, email

Description automatically generated

1. 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”.

A picture containing text

Description automatically generated

**Building Embedding Matrix for Training Dataset**

1. 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'.

Graphical user interface, text, application, email

Description automatically generated

**Generating 3 dimensional inputs for LSTM layers**

1. 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.

Graphical user interface, text, application, email, Teams

Description automatically generated

**For Validation and Test Dataset**

1. 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**

1. 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.

**A picture containing graphical user interface

Description automatically generated**

**Model Building**

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

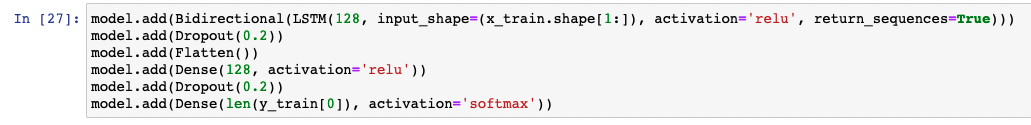
Graphical user interface

Description automatically generated with medium confidence

1. 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.



1. Compiling the model with Adam optimzizer and loss functiom of categorical\_crossentropy.

Graphical user interface, text, application

Description automatically generated

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

Table

Description automatically generated

**Model Evaluation**

1. 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.

Chart, line chart

Description automatically generated

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

Text

Description automatically generated with medium confidence

1. The model summary is displayed.

Table

Description automatically generated

# Model Testing

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

Text

Description automatically generated

1. 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”)

Text

Description automatically generated

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

Background pattern

Description automatically generated

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

Text

Description automatically generated

1. 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.

Table

Description automatically generated

**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).