Sentiment Analysis: CNN & LSTM report.

First, I loaded the JSON data and reformated it to make the JSON file format appropriate to convert it into a data frame.

Then we explore the columns of the data frame and check which features could help in our sentiment analysis. Sometimes, the features we get may not be the ones we need and so we would have to derive appropriate features according to our needs. Here, we use the "stars" feature to derive the sentiment.

We then map the number of stars with the sentiment. Essentially, we are quantifying the number of stars we get:

```
if stars_received <= 2:
        return -1
    elif stars_received == 3:
        return 0
    else:
        return 1
```

After the mapping, we study the distribution of the sentiment to better understand the Yelp review.

The three sentiments do not have an equal number of rows, and the issue of imbalanced classes will not be addressed in this post. Instead, a basic function has been created to extract a small number of records from each sentiment. Specifically, the top 10,000 records will be selected from each sentiment, resulting in a total of 30,000 records.

Although removing stop words would be a good start to preprocess the data in sentiment analysis sometimes stop words are necessary to determine if the text review is a positive review or a negative review. For instance:

Let the original text review be --> "I did not like the food!!"

On removing stop words we get --> "I like food!!"

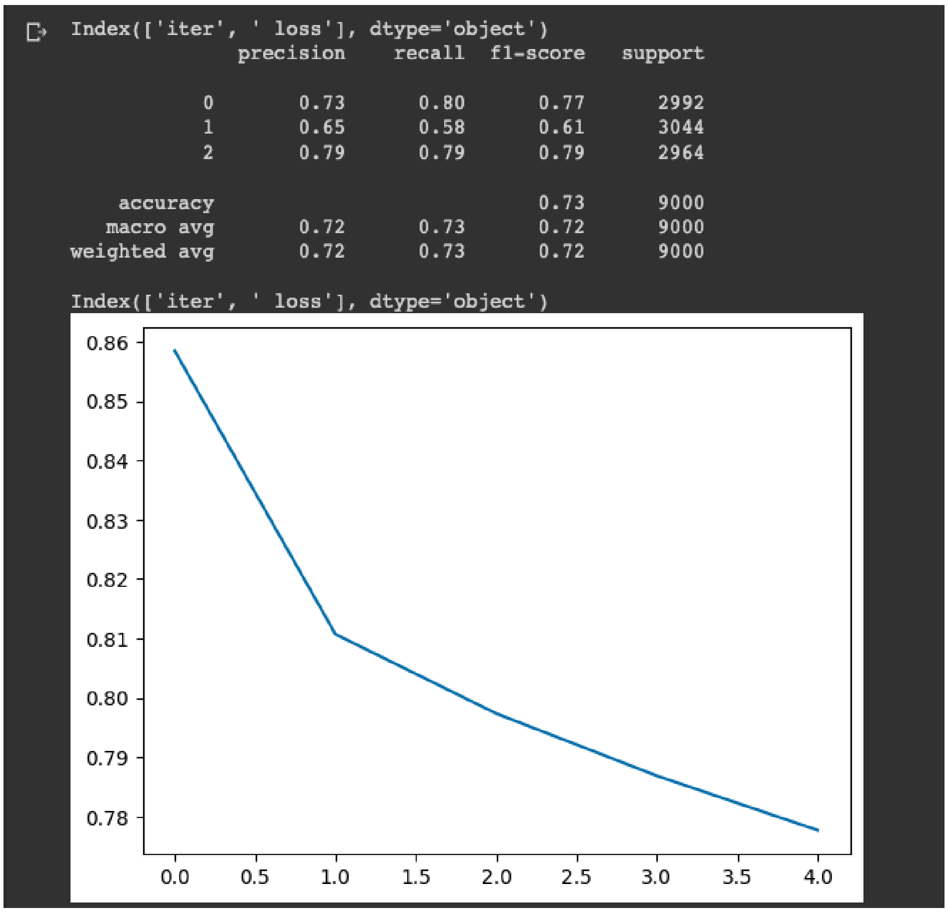
The 2 above sentences are complete opposites in terms of the sentiment of the sentence. Hence, we can avoid omitting the stop words in this problem.

The next logical preprocessing of the data would be Tokenization followed by stemming (that is, getting the word to its root form). Tokenization involves dividing a sentence or a piece of text into individual words or tokens, which enables separate transformations to be applied to each word. Additionally, tokenization is necessary to convert words into numerical representations.
Then the dataset is split into a 70-30 ratio (70 being the training set and 30 being the validation set). The idea is to maintain an equal distribution of classes in both sets to avoid biased outcomes or inadequate model training. This aspect is vital in machine learning models. In real-world scenarios, imbalanced classes can occur, requiring techniques such as oversampling the minority class or undersampling the majority class.

Now, let us dive into CNN. The first is the convolution layer. These layers aim to identify patterns by sliding a small kernel window across the input. Rather than applying filters to tiny sections of images, the window slides through the embedding vectors of a few words, as specified by the window size. To examine sequences of word embeddings, the window must encompass multiple word embeddings in a sequence. Consequently, the window dimensions will be rectangular, with a size of window_size * embedding_size. We will be using 10 filters of dimension 3 X 500. After passing the input through the filters we do max pooling. After obtaining the feature vector and extracting essential features, it is sufficient to recognize that a specific feature, such as a positive phrase like "great food," exists in the sentence without considering its position. Max pooling is employed to capture this information and discard the rest.
Note: The inputs were padded (window_size-1) to make the height of the same size.

Next, we train the model and see the accuracy of the validation set.
The model is trained for 5, 10, and 15 epochs with activation 'tanh' and 'relu'. The results are shown below.

An important thing to note here is that 1 epoch training time to train the CNN model was around 1 min 30 sec whereas for the LSTM it was around 10 minutes which is significantly higher. CNN takes way less time.



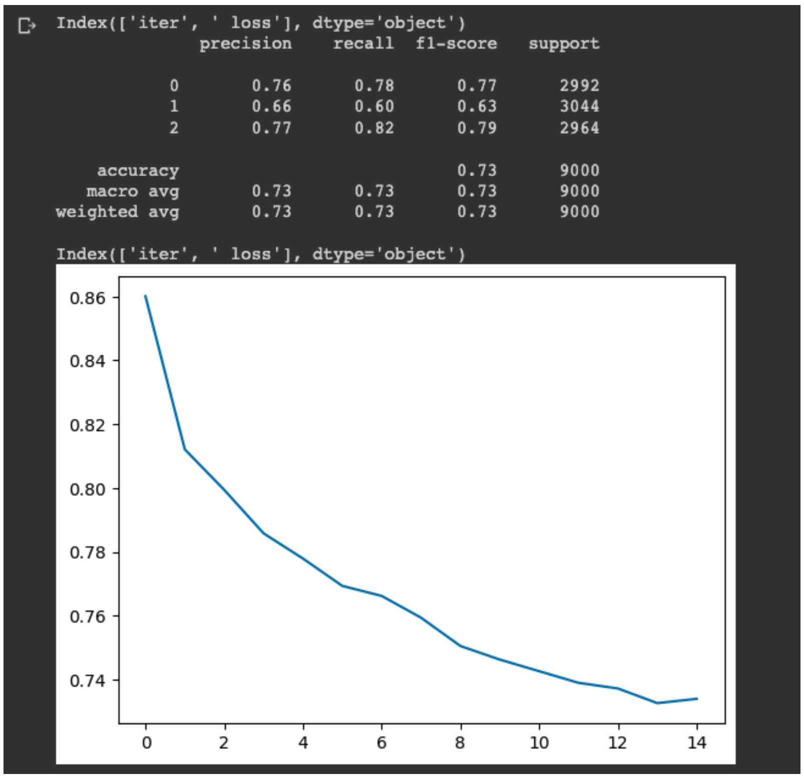
```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.73      0.80      0.77      2992
           1       0.65      0.58      0.61      3044
           2       0.79      0.79      0.79      2964

    accuracy                           0.73      9000
   macro avg       0.72      0.73      0.72      9000
weighted avg       0.72      0.73      0.72      9000

Index(['iter', ' loss'], dtype='object')
```

Image 1. Epoch = 5, Activation = tanh



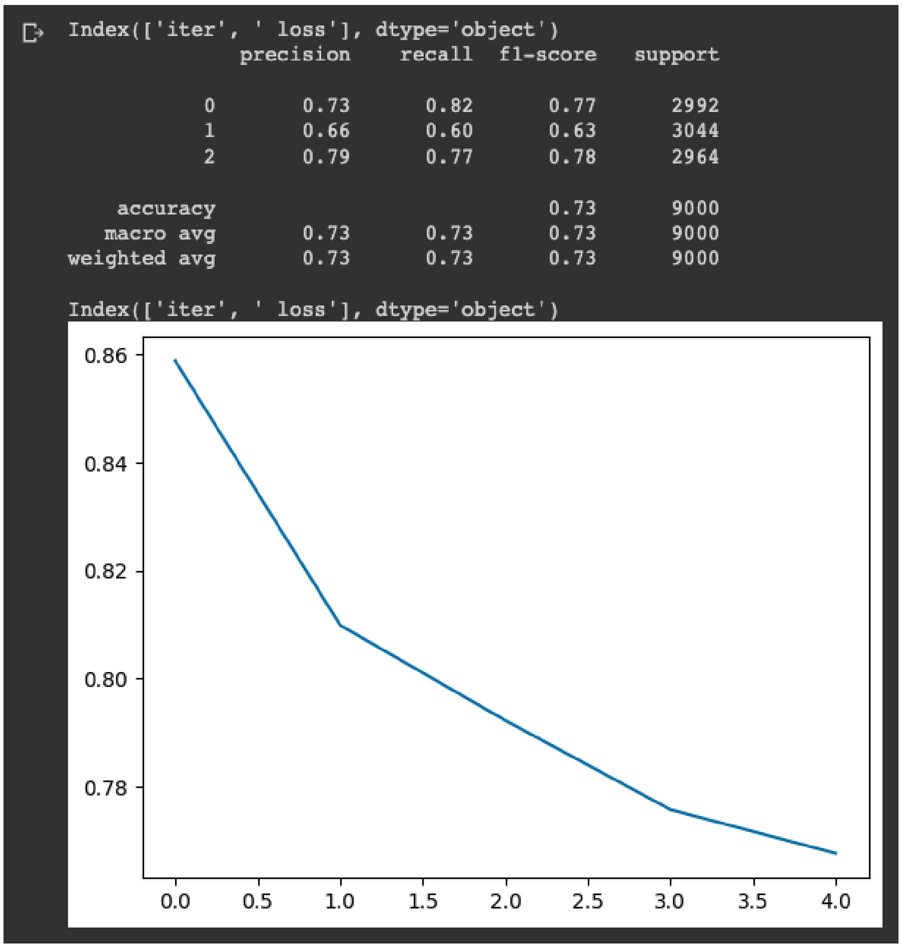
```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      2992
           1       0.66      0.60      0.63      3044
           2       0.77      0.82      0.79      2964

    accuracy                           0.73      9000
   macro avg       0.73      0.73      0.73      9000
weighted avg       0.73      0.73      0.73      9000

Index(['iter', ' loss'], dtype='object')
```

Image 2. Epoch = 15, Activation = tanh



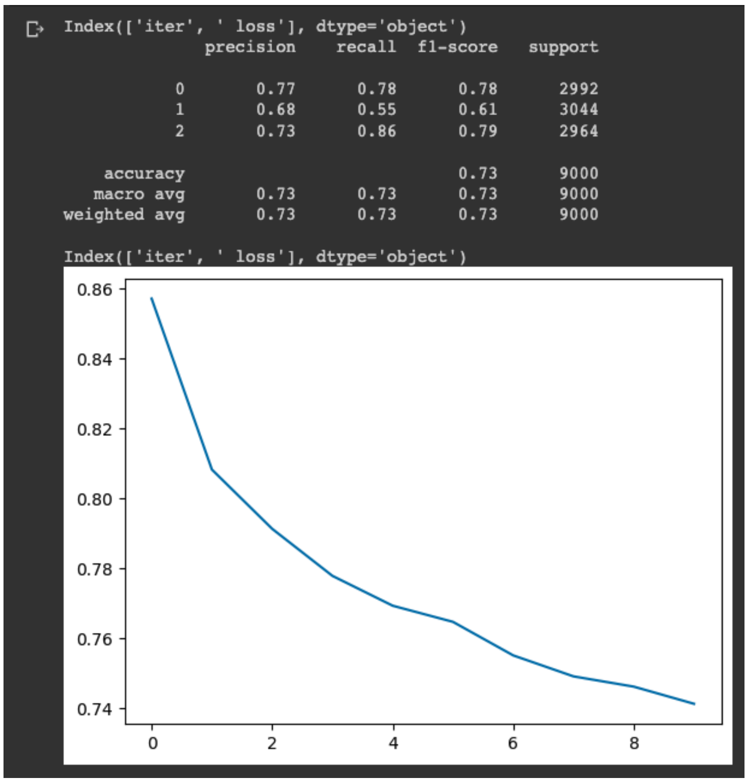
```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.73      0.82      0.77      2992
           1       0.66      0.60      0.63      3044
           2       0.79      0.77      0.78      2964

    accuracy                           0.73      9000
   macro avg       0.73      0.73      0.73      9000
weighted avg       0.73      0.73      0.73      9000

Index(['iter', ' loss'], dtype='object')
```

Image 3. Epoch = 5, Activation = relu



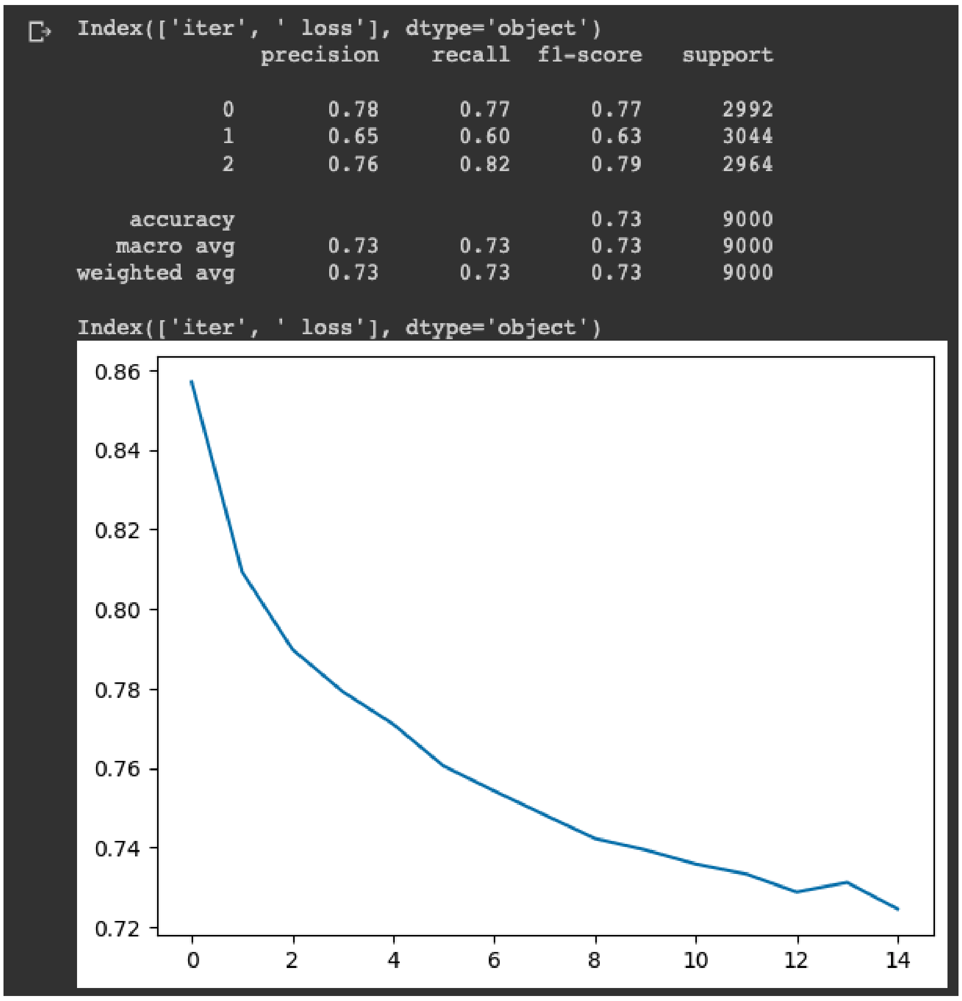
```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.77      0.78      0.78      2992
           1       0.68      0.55      0.61      3044
           2       0.73      0.86      0.79      2964

    accuracy                           0.73      9000
   macro avg       0.73      0.73      0.73      9000
weighted avg       0.73      0.73      0.73      9000

Index(['iter', ' loss'], dtype='object')
```

Image 4. Epoch = 10, Activation = relu



```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.78      0.77      0.77      2992
           1       0.65      0.60      0.63      3044
           2       0.76      0.82      0.79      2964

    accuracy                           0.73      9000
   macro avg       0.73      0.73      0.73      9000
weighted avg       0.73      0.73      0.73      9000

Index(['iter', ' loss'], dtype='object')
```

Image 5. Epoch = 15, Activation = relu

Although I did not observe any difference, in terms of accuracy, between the two activation functions but while using “relu” we can see there are 2 prominent inflection points in the graph shown in Image 3 as compared to Image 1 which uses “tanh”. Also, I observed something counterintuitive when comparing graphs in Image 2 and Image 5. In Image 5 we can clearly see the accuracy increased a bit between the 12th and 14th epochs forming a peak whereas in Image 2 there is a valley formed between the 12th and 14th epochs.

For the LSTM we do a slightly different data preprocessing, we consider two criteria. First, we exclude all reviews with 3 stars, as they are neutral, and our goal is to predict positive or negative sentiment. The second criterion involves maintaining an equal number of positive and negative reviews. We also take care of the imbalanced data and we separate the reviews and stars. We clean the data in a similar way as we did for CNN. For data cleaning, we undertake the following steps:

Eliminate punctuation, remove non-alphanumeric characters, divide each review into an array of words, and convert each word to lowercase.
Once the reviews are cleaned, we must generate embeddings for each review. Since we cannot directly feed the current text data into the model, we need to encode each word as a unique numerical value. As a result, a review will transform from an array of words to an array of integer values. To accomplish this, we can add an Embedding layer to the network, but we must first create the embedding map.

On further data analysis, it seems that the majority of reviews have a length of 20 to 40 words, with fewer reviews having lengths of 60, 80, and 100. Capping the word count at 60 seems to be a reasonable choice. As shown in the code below, we will employ the Keras function pad_sequences, which truncates longer reviews and pads shorter reviews with zeros, up to a maximum limit of 60 words.
Note: the total number of trainable parameters is 35,839,802 which is significantly larger than that of CNN.

The model features two LSTM layers, each containing 100 neurons, and does not use regularization. It is trained with an Adam optimizer and a batch size of 128.
The validation came out to be around 80% (I had to reduce the number of epochs as there was some weird error regarding the legacy Adam optimizer and Graph execution).

The accuracy is significantly better than the CNN model. The LSTM model I used is pretty basic and it can be improved by using better embeddings (like GloVe)