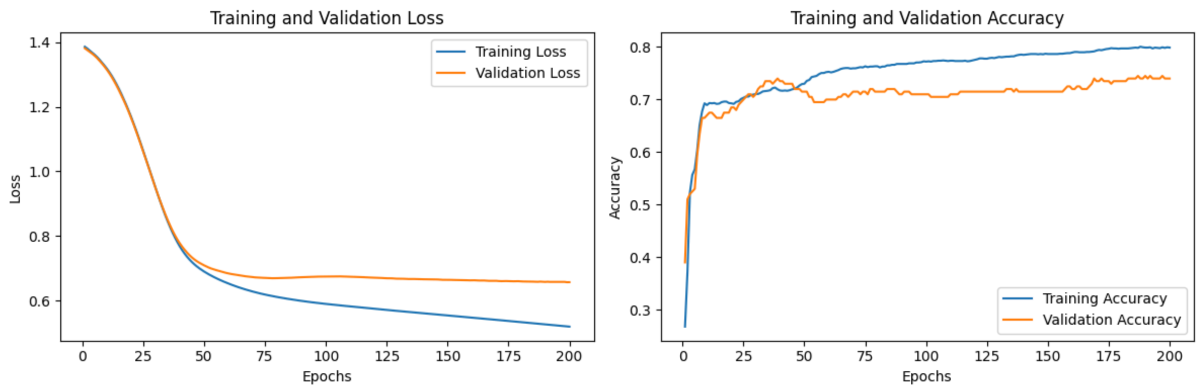
TEXT Classification using Word2Vec embeddings

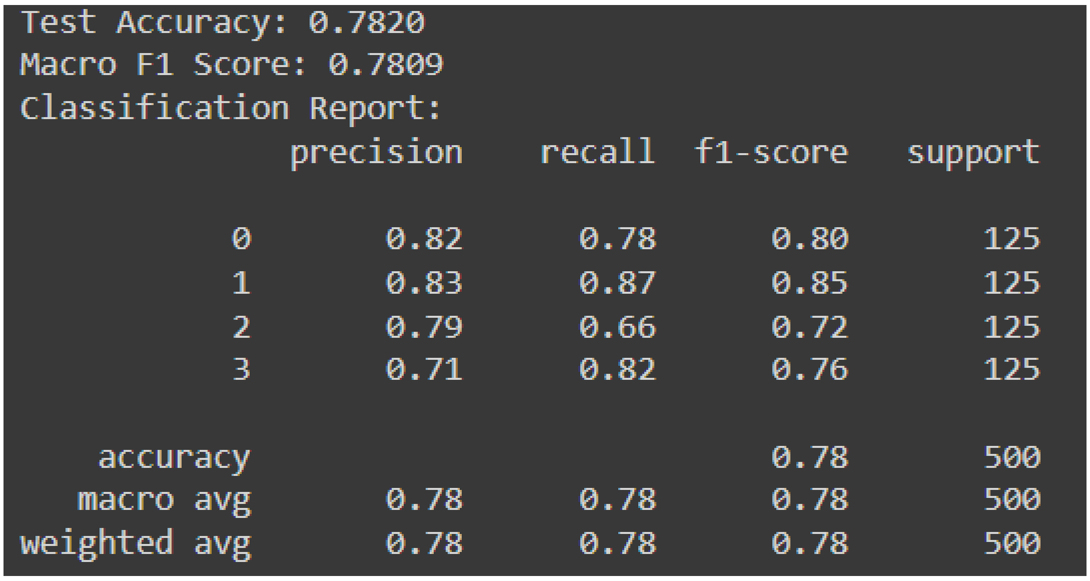
A) Word2Vec model with Neural Network:

1. First, all the important libraries are imported. Both train and test data are imported in the form of CSV and they are preprocessed which includes making the text lowercase, removing unnecessary symbols, and Replacing all \\ with " ". I have removed the stopwords using the nltk library.
2. Vocabulary is created using all the tokens from the train as well as the test dataset because we want that we should have vector embeddings of all the tokens. The total no. of tokens in the vocabulary that we have created is **14313.**
3. The max document frequency is **4336** for **"the"** token while the minimum document frequency is **1** for many tokens. One such token is "**frazzel"**.
4. I have used the word2vec model from the Python library genism. models and used the size of embeddings to be created as 200. This is chosen as I got the best model for this as well as it is greater than 138 which is the maximum no. of words in the sentence, so there is no loss of data. The model is trained on the vocabulary that I have created so that I can get word2vec embeddings for each token in the vocabulary.
5. After that, I created a NeuarNetwork class which will take average embeddings of sentences as input and return the label. It has 2 hidden layers having neurons 100 and 50 and finally, it will do the prediction about 4 classes.
6. A standard pytorch training loop is used and the best model is decided based on validation loss. I have tried out various combinations of hyperparameters like dropout rate, embedding size, learning rate, no. of hidden layers of the neural network, no. of epochs, and no. of neurons in hidden layers. I got the best result for the following set of hyperparameters:
   Learning rate: 0.001
   No. of hidden layers = 2
   Embedding size = 200
   Neurons in hidden layers = 100, 50
   No of epochs = 30
   Dropout rate = 0.5
7. The plots of training and validation loss vs. epochs as well as training and validation accuracy vs. epochs help us to visualize the training and decide the hyperparameter values like no. of epochs to set and the learning rate value of the Adam.



The plots show that there is no overfitting even after training for 200 epochs. The best model is obtained at **epoch 198**. Corresponding **Training Loss: 0.5203, Training Accuracy: 0.7983, Validation Loss: 0.6575, Validation Accuracy: 0.7400.** This best model is saved.

8. The saved best model is loaded and then tested on the test dataset where the results are as follows:

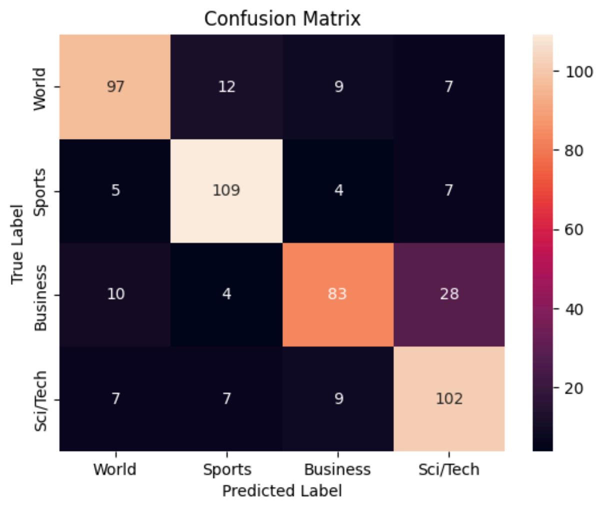


```
Test Accuracy: 0.7820
Macro F1 Score: 0.7809
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       125
           1       0.83      0.87      0.85       125
           2       0.79      0.66      0.72       125
           3       0.71      0.82      0.76       125

    accuracy                           0.78       500
   macro avg       0.78      0.78      0.78       500
weighted avg       0.78      0.78      0.78       500
```

The **test accuracy is 0.782** and the **f1-score is 0.780** which is good considering that training accuracy is 0.79. Also, the confusion matrix for this result is as follows:

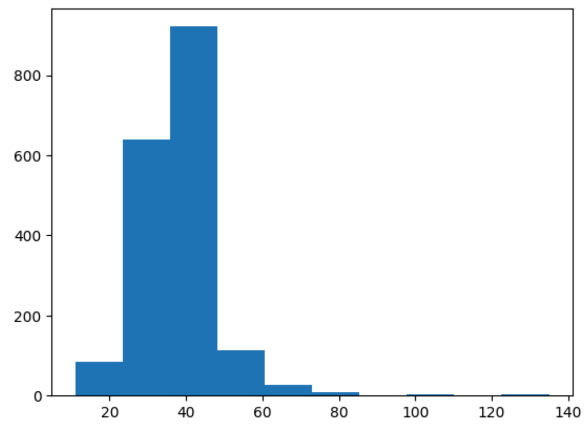


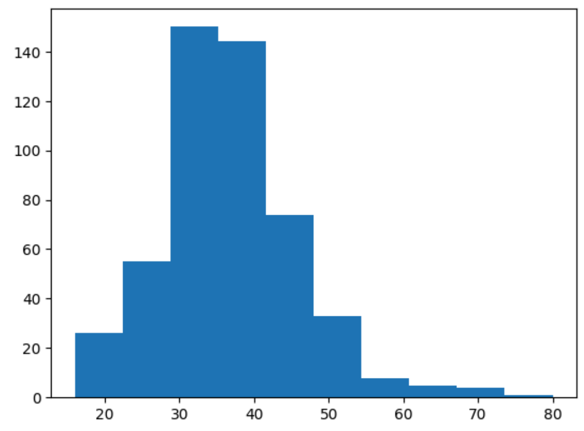
From the confusion matrix, we can see that many values are on the diagonal showing correct prediction. However, the incorrect predictions are also quite high and far from diagonals. It indicates that the model is making more errors in predicting those classes. For business class, many times it is predicted as Sci/Tech, which suggests that the model is misclassifying instances of that class more frequently.

B) Recurrent Neural Network:

Bi-directional RNN for Text classification:

1. Data is preprocessed the same as it was done before.
2. I have calculated the word count of each sentence in the training dataset to see the distribution of no. of words in the sentences and fix the maximum sequence length. The minimum, maximum, and average no. of words in a sentence in a training dataset is 11,135 and 38 while for the test dataset minimum, maximum, and average no. of words in a sentence in a test dataset is 16, 80, and 36. The distribution is as follows: (Left - Training Data, Right- Test Data)

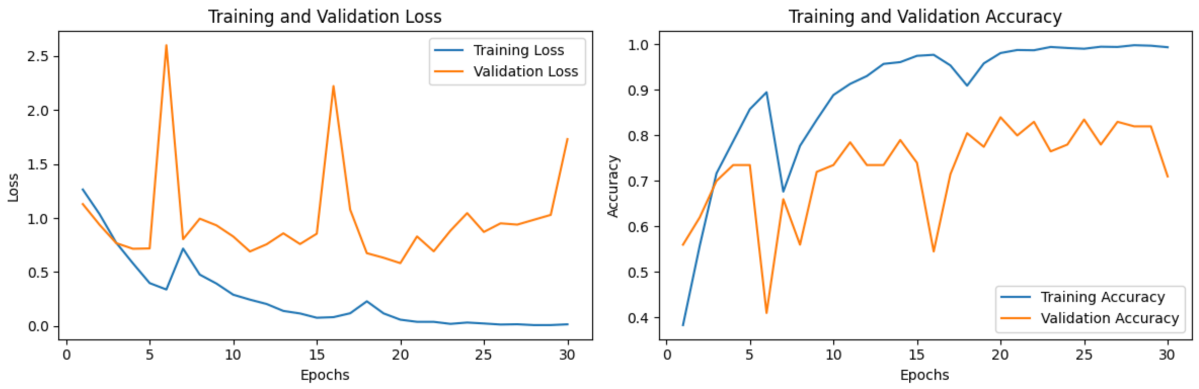




3. So, the maximum sequence length that I choose is 200, and based on this I have pad/truncated all the sentences in the train and test dataset.
4. I have used the same vocabulary created for part A, the only difference is that due to padding 1 extra token ‘<pad>’ was added to the vocabulary. After that, I mapped vocabulary to indices to tokenize them.
5. I have created an embedding matrix that stores embeddings of each of the tokens from the vocabulary. This is passed to the Embedding layer of the RNN model class.
6. A standard pytorch training loop is used and the best model is decided based on validation loss. I got the best result for the following set of hyperparameters:

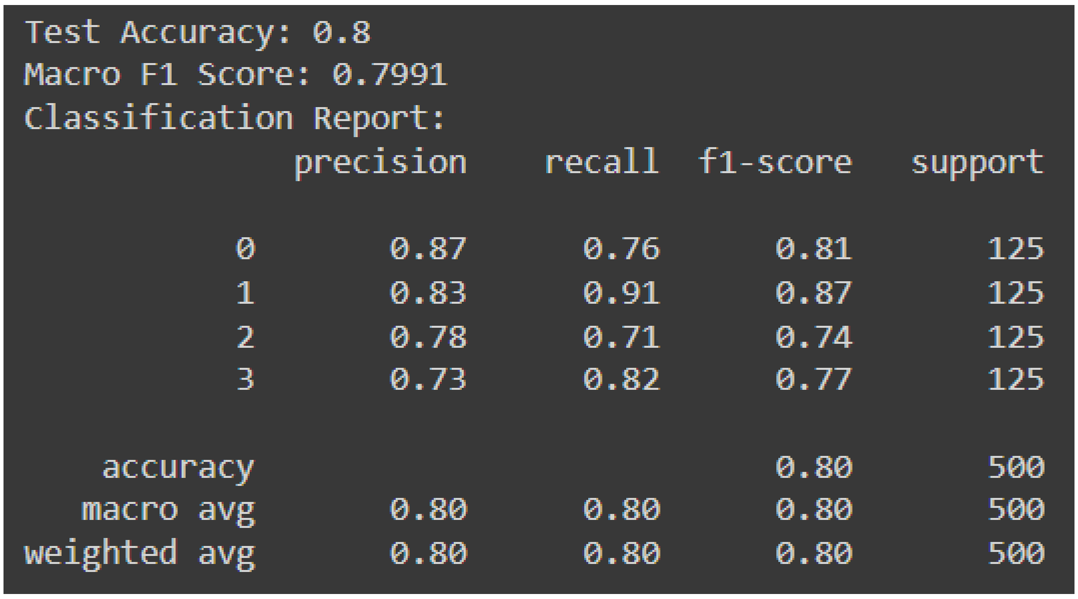
Learning rate: 0.001
No. of hidden layers = 2
Embedding size = 200
Neurons in hidden layers = 100, 50
No of epochs = 30
Dropout rate = 0.3
Num of layers of RNN = 2

7. The plots of training and validation loss vs. epochs as well as training and validation accuracy vs. epochs help us to visualize the training and decide the hyperparameter values like no. of epochs to set and the learning rate value of the Adam.



The plot shows that though validation loss does not have a smooth curve, it is decreasing. The best model is obtained at the **epoch 4**. Corresponding **Train Loss: 0.70581337, Train Accuracy: 0.68222222, Val Loss: 0.81090186, Val Accuracy: 0.74000000.** This best model is saved.

8. The saved best model is loaded and then tested on the test dataset where the results are as follows:



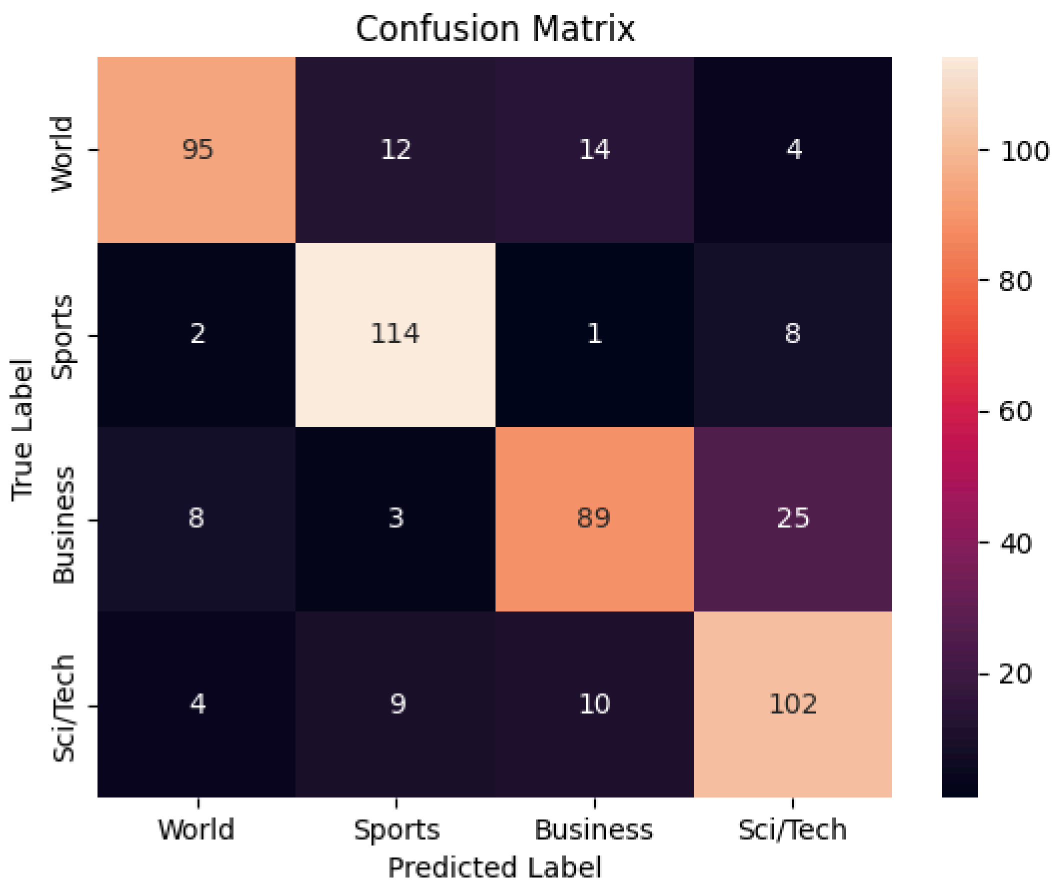
```
Test Accuracy: 0.8
Macro F1 Score: 0.7991
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.76      0.81       125
           1       0.83      0.91      0.87       125
           2       0.78      0.71      0.74       125
           3       0.73      0.82      0.77       125

    accuracy                           0.80       500
   macro avg       0.80      0.80      0.80       500
weighted avg       0.80      0.80      0.80       500
```

The **test accuracy is 0.8** and the **f1-score is 0.799** which is good considering that training accuracy is 0.79

9. The confusion matrix for the test dataset is as follows:



True \ Predicted	World	Sports	Business	Sci/Tech
World	95	12	14	4
Sports	2	114	1	8
Business	8	3	89	25
Sci/Tech	4	9	10	102

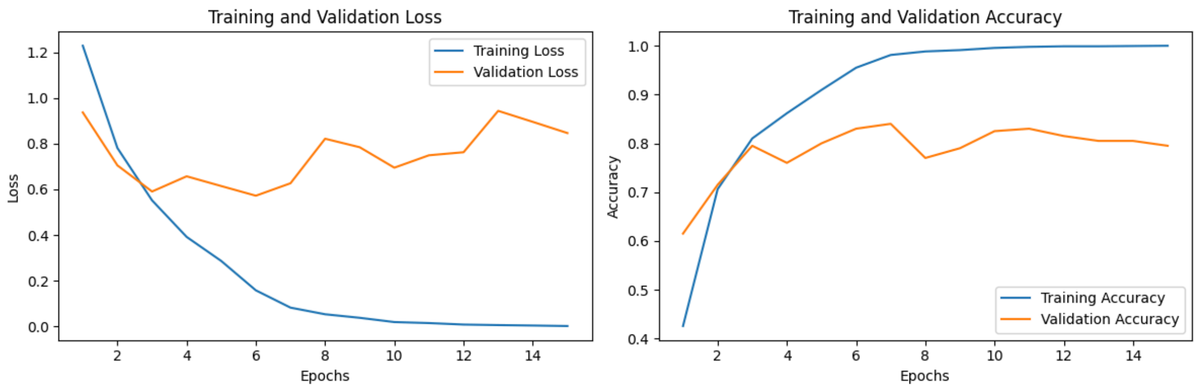
From, the confusion matrix, it can be seen that again many values are on the diagonal showing correct prediction. Also, the no. of entries on the diagonal is more than the case of RNN. as well as no. of incorrect predictions is reduced. It indicates that we have trained a better model as compared to the normal neural network.

Bi-directional LSTM for Text classification:

1. For Bi-directional LSTM, I have defined a separate model class and all other things remain the same as for bi-directional RNN.
2. A standard pytorch training loop is used and the best model is decided based on validation loss. I got the best result for the following set of hyperparameters:
   Learning rate: 0.001
   No. of hidden layers = 2
   Embedding size = 200
   Neurons in hidden layers = 100, 50

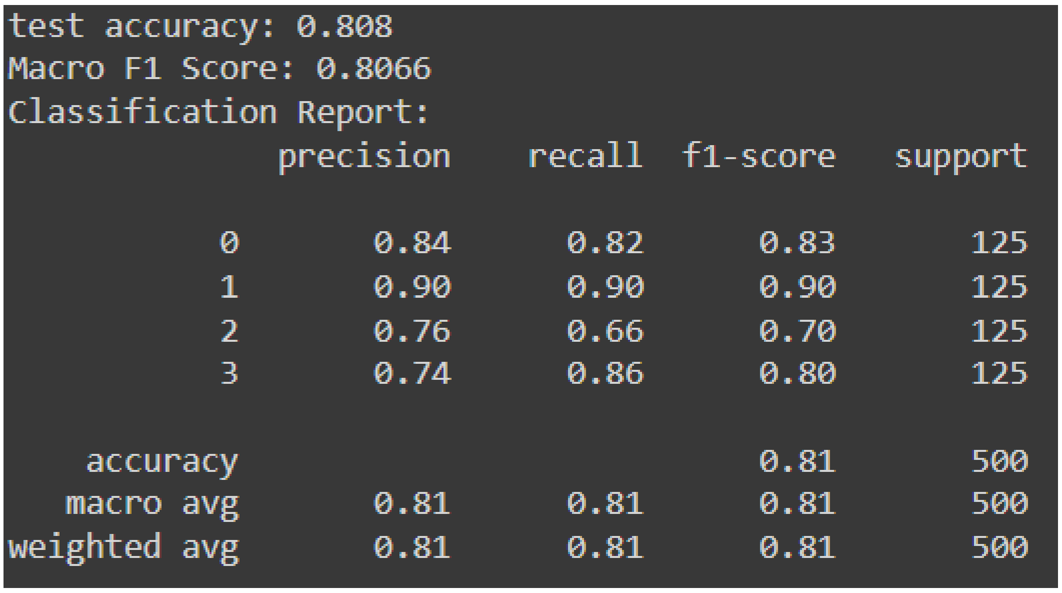
No of epochs = 15
Dropout rate = 0.4
Num of layers of LSTM = 1

3. The plots of training and validation loss vs. epochs as well as training and validation accuracy vs. epochs help us to visualize the training and decide the hyperparameter values like no. of epochs to set and the learning rate value of the Adam.



The best model is found at **epoch 5. Train Loss: 0.28632774, Train Accuracy: 0.90944444, Val Loss: 0.61457224, Val Accuracy: 0.80000000.** The training accuracy also reaches 1 after certain epochs which shows that if we train further, the model will overfit.

4. The saved best model is loaded and then tested on the test dataset where the results are as follows:



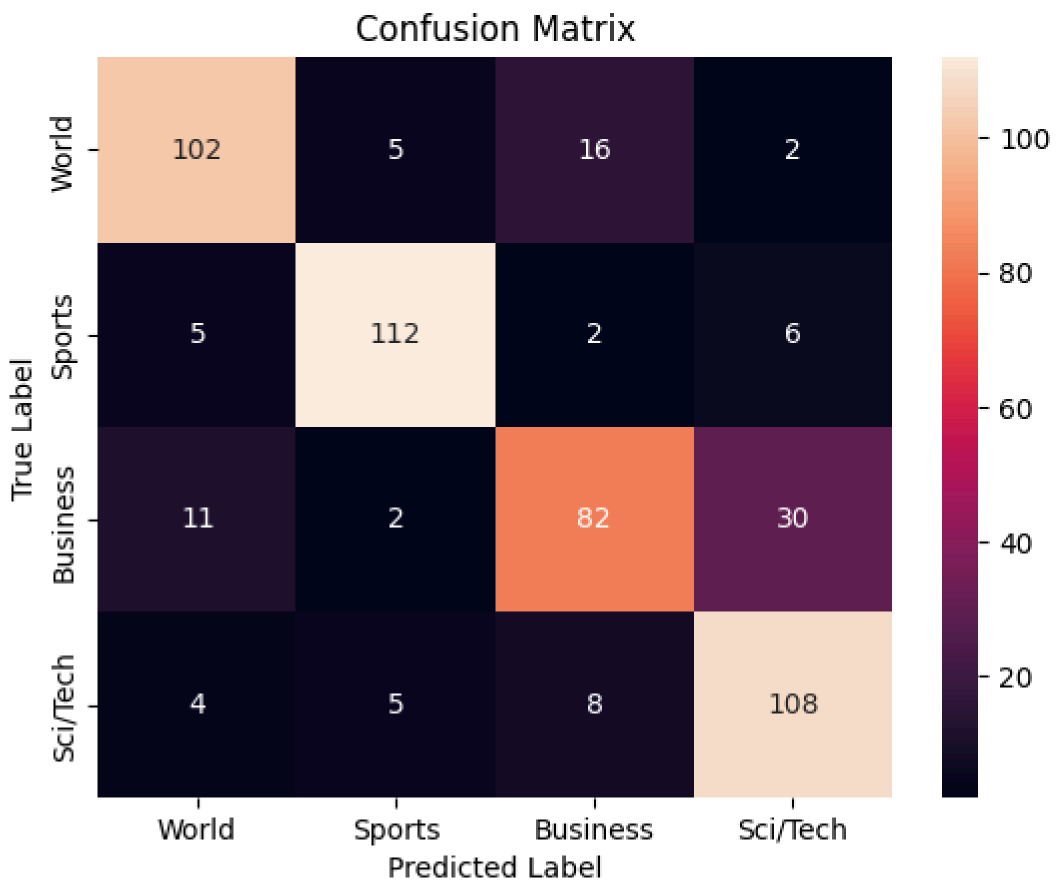
```
test accuracy: 0.808
Macro F1 Score: 0.8066
Classification Report:
             precision    recall  f1-score   support

           0       0.84      0.82      0.83       125
           1       0.90      0.90      0.90       125
           2       0.76      0.66      0.70       125
           3       0.74      0.86      0.80       125

    accuracy                           0.81       500
   macro avg       0.81      0.81      0.81       500
weighted avg       0.81      0.81      0.81       500
```

The **test accuracy is 0.808** and the **f1-score is 0.8066** which is good considering that training accuracy is 0.90

5. The confusion matrix is as follows:



True \ Predicted	World	Sports	Business	Sci/Tech
World	102	5	16	2
Sports	5	112	2	6
Business	11	2	82	30
Sci/Tech	4	5	8	108

From, the confusion matrix, it can be seen that again many values are on the diagonal showing correct prediction. Also, the no. of entries on the diagonal is more than the case of RNN. as well as no. of incorrect predictions is reduced. It indicates that we have trained a better model as compared to the normal neural network.

We can see that we are getting better results as we go from neural network to Bidirectional-RNN, Bidirectional-LSTM.

Challenges Faced:

1. The documentation of the genism library was not good and it was not given how we have to train the word2vec model. So, I have just applied the model and thought that embeddings are generated though they are not in reality as the model is not trained. So embeddings were just random no. and when I used this embedding my accuracy was stuck in between 35-40%. And it was increasing further. I later realized that I had to train the word2vec model also when I went through one of the articles of Analytics Vidhya