Report

Name: Riya Paradkar

Date: 27th May 2021

Task: Create a classification model based on the given dataset.

Dataset:https://docs.google.com/spreadsheets/d/1DLL6BTXiHHsn1w9NvVi0BZbass0QU0RSvKEXtJlwfCM/

It consist of

Text: contains text from block chain domain

Target: target class

Approach to solve the problem:

I have first imported the data. Then dropped the null entries from given dataset. As the dataset consist of sentences or strings, I have applied label encoding to the target to get some valuable insights from the model and it will be easier to work with numbers rather than strings. Using Label encoding target was labelled from 0-10 i.e. 11 targets.

After that the whole dataset was split into test and train.

The input data consists of sentences. The labels to predict are either 0 or 1.

One way to represent the text is to convert sentences into embedding's vectors. We can use a pre-trained text embedding as the first layer, which will have two advantages:

- We don't have to worry about text preprocessing,
- We can benefit from transfer learning.

Then first I created a Keras layer that uses a TensorFlow Hub model to embed the sentences. Then build the full model using model (https://tfhub.dev/google/nnlm-en-dim128-with-normalization/2")

In this model the layers are stacked sequentially to build the classifier:

- The first layer is a TensorFlow Hub layer.
- This layer uses a pre-defined Saved Model to map a sentence into its embedding vector. The model splits the sentence into tokens, embeds each token and then combines the embedding. This fixed-length output vector is piped through a fully-connected layer with 16 hidden units.
- The last layer used is softmax layer.

Report

```
In [11]: model = tf.keras.Sequential()
         model.add(hub_layer)
         # model.add(tf.keras.layers.Dense(50,input_shape=(1,50), activation='relu'))
         model.add(tf.keras.layers.Dropout(0.25))
         model.add(tf.keras.layers.Dense(16, activation='relu'))
         model.add(tf.keras.layers.Dropout(0.2))
         model.add(tf.keras.layers.Dense(11,activation = 'softmax'))

         model.summary()
```

A model needs a loss function and an optimizer for training. I have used Crossentropy loss function as there are two labeled class. To provide labels using one-hot representation, hence SparseCategoricalCrossentropy loss function is used.

```
In [12]: model.compile(optimizer='adam',
                       loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
                       metrics=[tf.metrics.SparseCategoricalAccuracy(name='accuracy')])
         model.fit(X_train, y_train, epochs=3, batch_size = 32)

         Epoch 1/3
```

All the weights are saved in drive file. Then finally I have predicted the results using test data and accuracy was found 68.67%.

Model Interpretation:

From the accuracy scores we can observe that the model performs better on the training data than it does on data it has never seen before i.e. Test Data.

Train & Test accuracy score:

Train accuracy score=0.7151 i.e. 71.5%

```
         Epoch 3/3
         476/476 [==============================] - 593s 1s/step - loss: 0.8918 - accuracy: 0.7151
Out[12]: <tensorflow.python.keras.callbacks.History at 0x7f982baf3610>
```

Test accuracy score: 0.6867325146823278 i.e.68.67%

Report

```
In [16]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,y_pred)

Out[16]: 0.6867325146823278
```

Limitation of the model:

- We tried training it for more epoch the training accuracy increased but the testing accuracy decreased, the model was over fitting.
- Drop out layer was also applied but still it was little over fitting.
- It does not give good prediction on test data.
- Training it fir more epochs made it over fit on training data.
- It maybe requires lot of data to get train and give much better accuracy score.