Lab 11 – Supervised Artificial Neural Network for Intrusion Detection

Dr. Sana Ullah Jan

- **Objective:** Implement and analyse performance of Supervised Artificial Neural Network (ANN) for intrusion detection.
- **Dataset:** A subset of CIC-IDS2017 dataset will be provided to students (available on https://www.unb.ca/cic/datasets/ids-2017.html [Accessed on 27/10/2021]).
- **Exercise:** Vary the input parameters of ANN to see the effect on its performance.
- **Platform:** Google Colab (https://colab.research.google.com).

Introduction

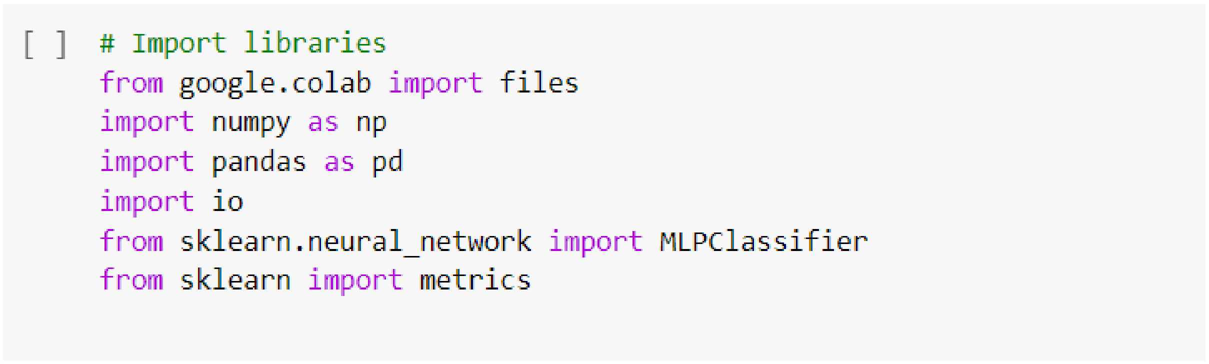
This lab will present implementation and performance analysis of Artificial Neural Networks (ANN) used for classification and identification of intrusion. We will briefly cover simple implementations of ANN with Python as well as convenient ways of validating them using tuning functions.

Artificial Neural Networks

Completely covering neural networks would take us a complete term (and we would still be missing a lot!). This lab will only present a simple way of building a neural network-based classifier using the *sklearn.neural_network.MLPClassifier* library. We will use a small dataset of CIC-IDS2017 dataset for this problem.

Importing Necessary Libraries

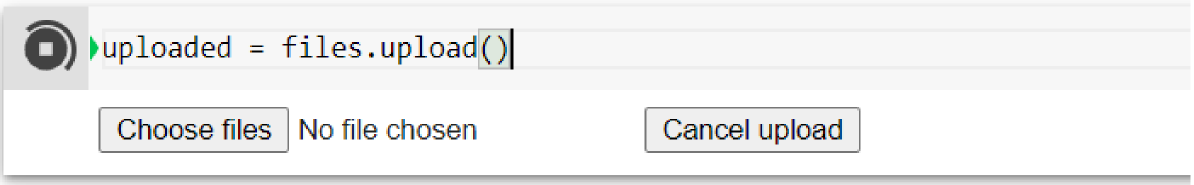
The necessary libraries are imported to the Python environment as follows



```
[ ] # Import libraries
    from google.colab import files
    import numpy as np
    import pandas as pd
    import io
    from sklearn.neural_network import MLPClassifier
    from sklearn import metrics
```

Uploading data

To upload data to the Google Colab environment, files.upload() function is used. Running it will generate an option to choose the files that you want to upload.

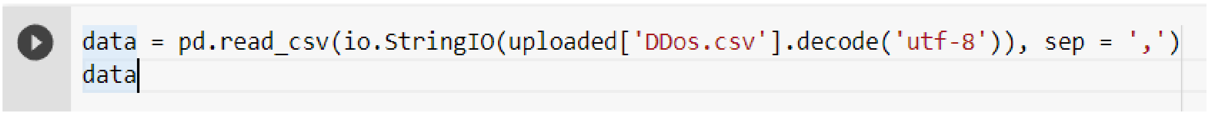


```
uploaded = files.upload()
```

Click on the “Choose files” to navigate to the location of the data “DDos”.

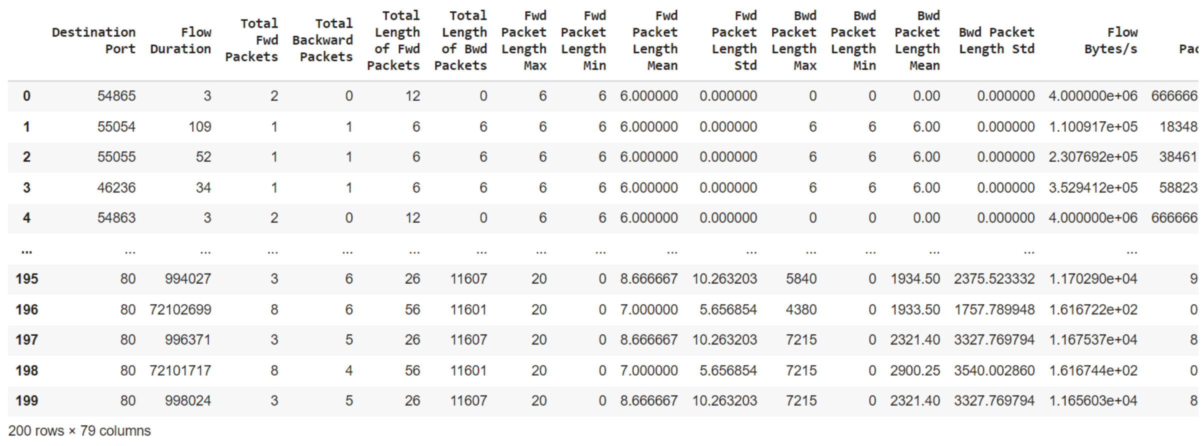
Reading and Decoding data

The data is uploaded as ‘Dictionary’ type to the Python environment with on entry named ‘DDos.csv’. To change this data to a DataFrame, run the following command



```
data = pd.read_csv(io.StringIO(uploaded['DDos.csv'].decode('utf-8')), sep = ',')
data
```

This will show the data as follows

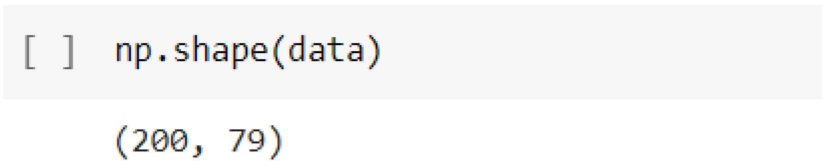


	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	Bwd Packet Length Min	Bwd Packet Length Mean	Bwd Packet Length Std	Flow Bytes/s	Pac
0	54865	3	2	0	12	0	6	6	6.000000	0.000000	0	0	0.00	0.000000	4.000000e+06	666666
1	55054	109	1	1	6	6	6	6	6.000000	0.000000	6	6	6.00	0.000000	1.100917e+05	18348
2	55055	52	1	1	6	6	6	6	6.000000	0.000000	6	6	6.00	0.000000	2.307692e+05	38461
3	46236	34	1	1	6	6	6	6	6.000000	0.000000	6	6	6.00	0.000000	3.529412e+05	58823
4	54863	3	2	0	12	0	6	6	6.000000	0.000000	0	0	0.00	0.000000	4.000000e+06	666666
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
195	80	994027	3	6	26	11607	20	0	8.666667	10.263203	5840	0	1934.50	2375.523332	1.170290e+04	9
196	80	72102699	8	6	56	11601	20	0	7.000000	5.656854	4380	0	1933.50	1757.789948	1.616722e+02	0
197	80	996371	3	5	26	11607	20	0	8.666667	10.263203	7215	0	2321.40	3327.769794	1.167537e+04	8
198	80	72101717	8	4	56	11601	20	0	7.000000	5.656854	7215	0	2900.25	3540.002860	1.616744e+02	0
199	80	998024	3	5	26	11607	20	0	8.666667	10.263203	7215	0	2321.40	3327.769794	1.165603e+04	8

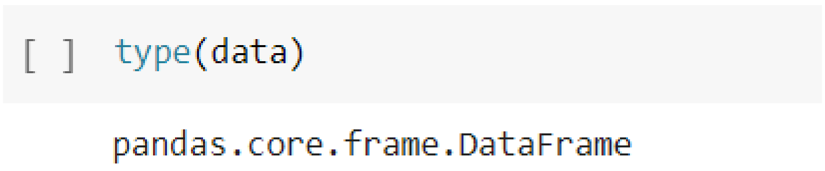
200 rows × 79 columns

Analysing data

Following commands are used to check the size and type of data



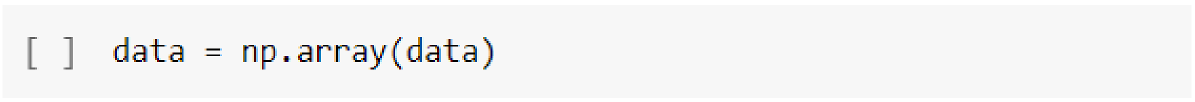
```
[ ]  np.shape(data)

     (200, 79)
```



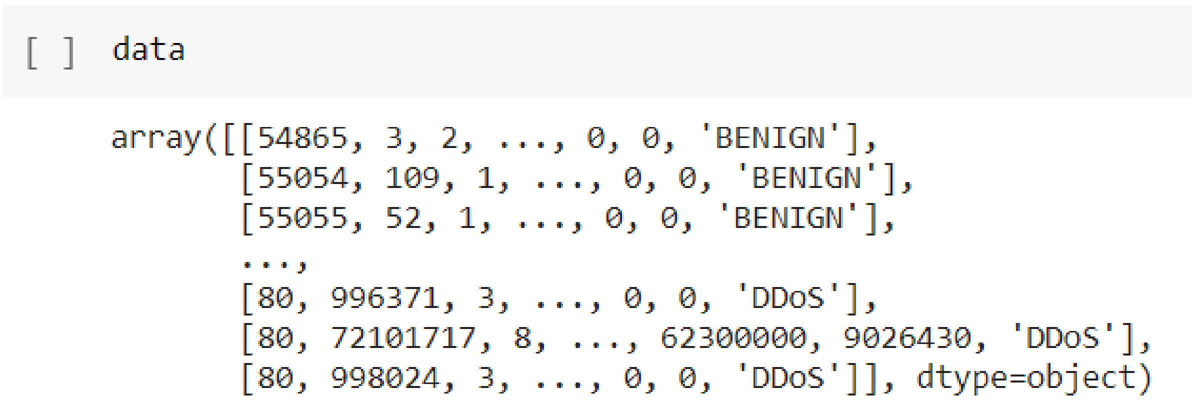
```
[ ]  type(data)

     pandas.core.frame.DataFrame
```

Changing data type

The data should be changed to array type in order to use it



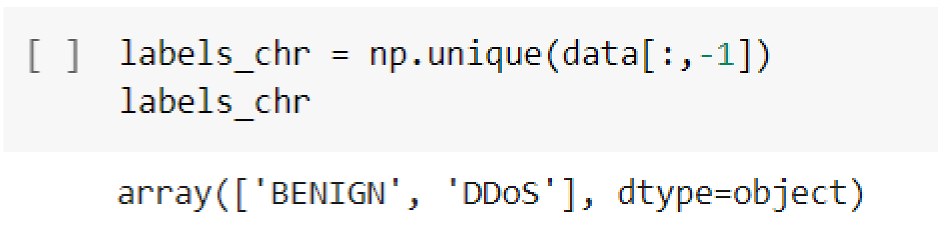
```
[ ]  data = np.array(data)
```



```
[ ]  data

     array([[54865, 3, 2, ..., 0, 0, 'BENIGN'],
            [55054, 109, 1, ..., 0, 0, 'BENIGN'],
            [55055, 52, 1, ..., 0, 0, 'BENIGN'],
            ...,
            [80, 996371, 3, ..., 0, 0, 'DDoS'],
            [80, 72101717, 8, ..., 62300000, 9026430, 'DDoS'],
            [80, 998024, 3, ..., 0, 0, 'DDoS']], dtype=object)
```

Labels of data

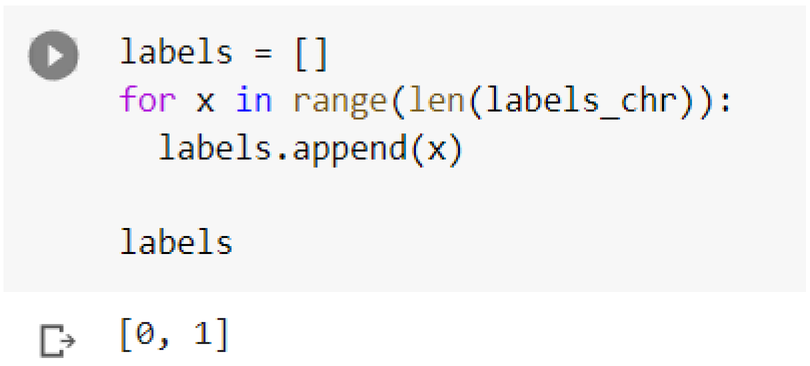
Obtain the list of labels given in the data. In this case, only two labels are given including ‘BENIGN’ and ‘DDoS’ which is the intrusion attempt.



```
[ ]  labels_chr = np.unique(data[:,-1])
     labels_chr

     array(['BENIGN', 'DDoS'], dtype=object)
```

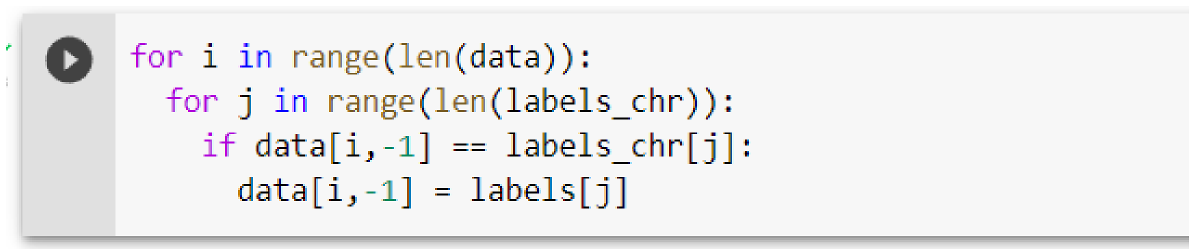
Enumerating the labels

As the ANN takes labels in form numbers 0,1,2,…; replace the labels with numbers where each label takes a number. In this case, ‘BENIGN’ is replaced with 0 and ‘DDoS’ is replaced with 1.

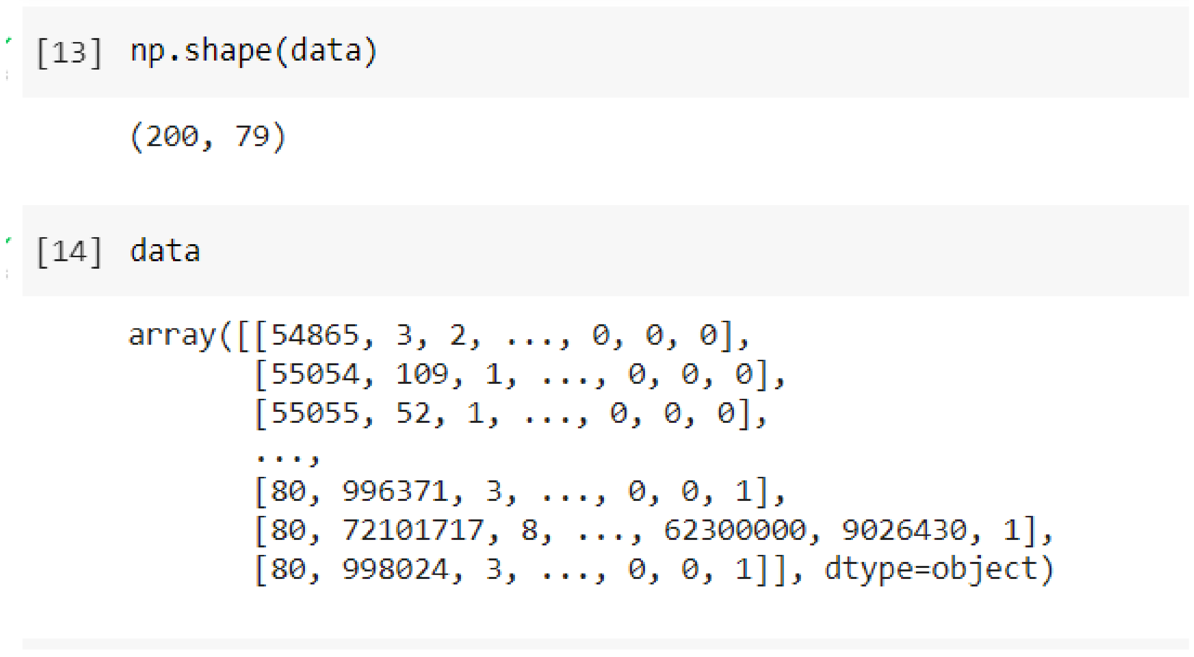


```
labels = []
for x in range(len(labels_chr)):
  labels.append(x)

labels

[0, 1]
```

Now, transform this into the data as follows



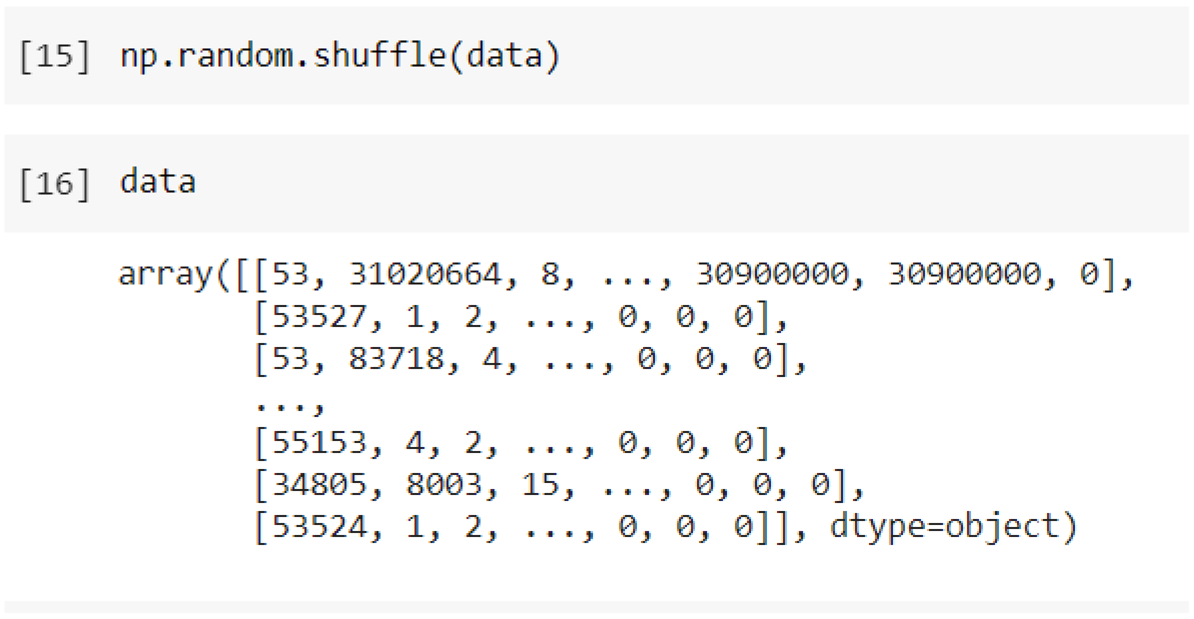
```
for i in range(len(data)):
  for j in range(len(labels_chr)):
    if data[i,-1] == labels_chr[j]:
      data[i,-1] = labels[j]
```



```
[13] np.shape(data)

     (200, 79)

[14] data

     array([[54865, 3, 2, ..., 0, 0, 0],
            [55054, 109, 1, ..., 0, 0, 0],
            [55055, 52, 1, ..., 0, 0, 0],
            ...,
            [80, 996371, 3, ..., 0, 0, 1],
            [80, 72101717, 8, ..., 62300000, 9026430, 1],
            [80, 998024, 3, ..., 0, 0, 1]], dtype=object)
```

Shuffle data

Shuffle the data to mix samples from both classes. Shuffling is also important to increase the generalization capability of the classifier. Also, It may help reduce biasness which ultimately help reduce the chances of overfitting and underfitting classifier.



```
[15] np.random.shuffle(data)

[16] data

     array([[53, 31020664, 8, ..., 30900000, 30900000, 0],
            [53527, 1, 2, ..., 0, 0, 0],
            [53, 83718, 4, ..., 0, 0, 0],
            ...,
            [55153, 4, 2, ..., 0, 0, 0],
            [34805, 8003, 15, ..., 0, 0, 0],
            [53524, 1, 2, ..., 0, 0, 0]], dtype=object)
```

Training and Testing Subsets

Now that data is preprocessed and ready for being used to train the classifier, we divide data into two subsets including train_data and test_data with respective train_labels and test_labels. Here, train_size represent the percent of data utilized in training. In this case, train_size = 0.8 means that 80% of the data is used training while remaining 20% data is used for testing or analysing the performance of classifier.

```
[17] train_size = 0.8
     no_train_samples = int(round(len(data)*train_size))
     train_data = data[0:no_train_samples,0:np.shape(data)[1]-1]
     train_labels = data[0:no_train_samples,-1].tolist()
     test_data = data[no_train_samples:,0:np.shape(data)[1]-1]
     test_labels = data[no_train_samples:,-1].tolist()
```

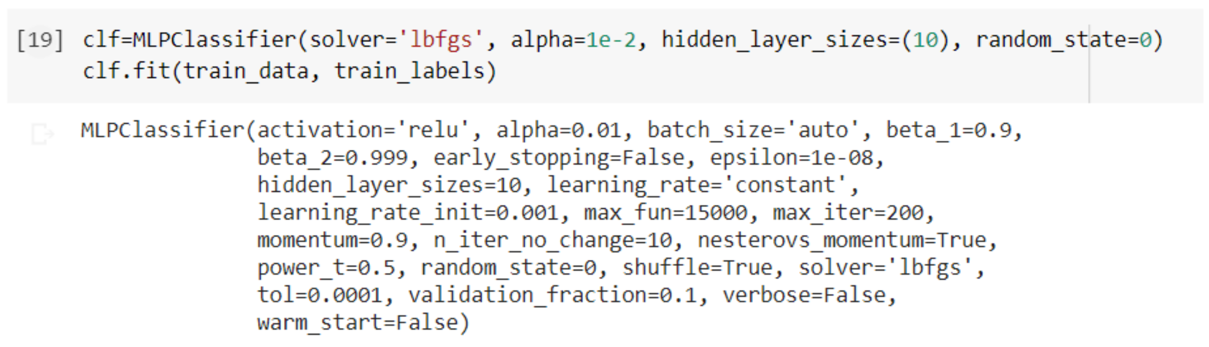
```
[18] train_data[train_data == np.nan]=0
     train_data[train_data == np.inf]=0
     test_data[test_data == np.nan]=0
     test_data[test_data == np.inf]=0
```

Train Classifier

The train_data along with train_labels is used to train the classifier. The parameters used are

- **solver:** {‘lbfgs’, ‘sgd’, ‘adam’}, default=’adam’
  The solver for weight optimization.
  - ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
  - ‘sgd’ refers to stochastic gradient descent.
  - ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

  Note: The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better.
- **alpha** : float, default=0.0001
  L2 penalty (regularization term) parameter.
- **Hidden_layer_sizes:** tuple, length = n_layers - 2, default=(100,)
  The ith element represents the number of neurons in the ith hidden layer.

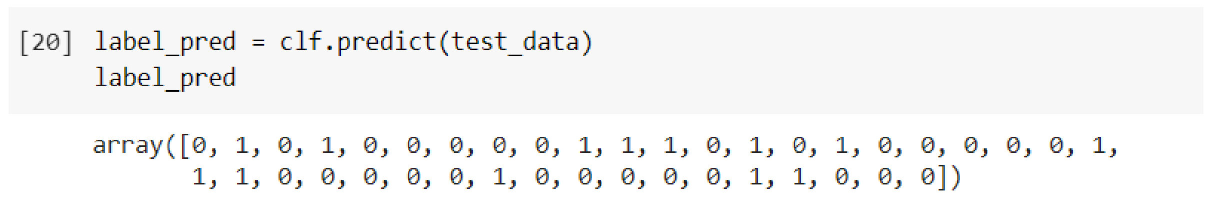


```
[19] clf=MLPClassifier(solver='lbfgs', alpha=1e-2, hidden_layer_sizes=(10), random_state=0)
     clf.fit(train_data, train_labels)
```

```
MLPClassifier(activation='relu', alpha=0.01, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=0, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

Prediction with Trained Classifier

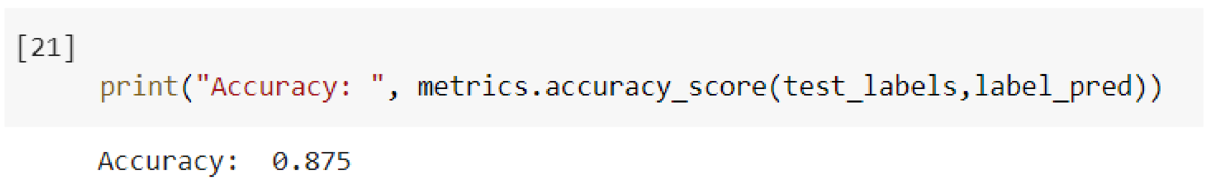
Now, its time to assess the performance of trained classifier using the test_data



```
[20] label_pred = clf.predict(test_data)
     label_pred
```

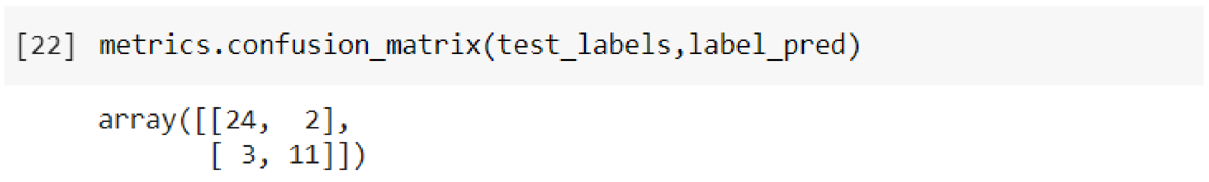
```
array([0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0])
```

Performance Assessment



```
[21]
     print("Accuracy: ", metrics.accuracy_score(test_labels,label_pred))
```

```
Accuracy:  0.875
```



```
[22] metrics.confusion_matrix(test_labels,label_pred)
```

```
array([[24,  2],
       [ 3, 11]])
```

Exercise

Change the following values to see its effect on performance

Solver = ‘sgd’, ‘adam’

Alpha = 1e-1, 1e-3, 1e-4

Hidden_layer_sizes = [20], [40, 20], [60, 40, 20]

Train_size = 0.5, 0.6, 0.7, 0.9