

# Forward School

**Program Code: J620-002-4:2020**

**Program Name: FRONT-END SOFTWARE DEVELOPMENT**

**Title : Exe29 - Neural Network Exercise 1**

**Name: Chong Mun Chen**

**IC Number: 960327-07-5097**

**Date : 31/7/2023**

**Introduction : Practising on this exercise with Neural Network learning model.**

**Conclusion : Succeeded in training a Multi-Layer Perceptron Classifier with a high accuracy score.**

## Neural Network Introduction

This exercise is adapted from <https://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html> (<https://www.kdnuggets.com/2016/10/beginners-guide-neural-networks-python-scikit-learn.html>)

We'll use SciKit Learn's built in Breast Cancer Data Set which has several features of tumors with a labeled class indicating whether the tumor was Malignant or Benign. We will try to create a neural network model that can take in these features and attempt to predict malignant or benign labels for tumors it has not seen before. Let's go ahead and start by getting the data!

```
In [1]: from sklearn.datasets import load_breast_cancer  
  
data = load_breast_cancer()
```

This object is like a dictionary, it contains a description of the data and the features and targets:

In [2]: *# find out the attributes in the dataset*

```
data.keys()
```

Out[2]: dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

In [3]: *# find out the total instances and number of features*

```
data.data.shape
```

Out[3]: (569, 30)

Set up the data (x) and labels (y)

In [4]: `X = data.data`  
`y = data.target`

### Train Test Split

Let's split our data into training and testing sets, this is done easily with SciKit Learn's `train_test_split` function from `model_selection`:

In [5]: `from sklearn.model_selection import train_test_split`  
`X_train, X_test, y_train, y_test = train_test_split(X, y)`

### Data Preprocessing

The neural network may have difficulty converging before the maximum number of iterations allowed if the data is not normalized. Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data. Note that you must apply the same scaling to the test set for meaningful results. There are a lot of different methods for normalization of data, we will use the built-in `StandardScaler` for standardization.

In [6]: *# Import the StandardScaler library*  
`from sklearn.preprocessing import StandardScaler`  
`sc = StandardScaler()`  
*# Fit only to the training data*  
`sc.fit(X_train)`

Out[6]: 

▼ StandardScaler

  

StandardScaler()

```
In [7]: # Now apply the transformations to the data:
X_train_transformed = sc.transform(X_train)
X_test_transformed = sc.transform(X_test)
```

## Training the model

Now it is time to train our model. SciKit Learn makes this incredibly easy, by using estimator objects. In this case we will import our estimator (the Multi-Layer Perceptron Classifier model) from the `neural_network` library of SciKit-Learn!

```
In [8]: from sklearn.neural_network import MLPClassifier
```

Next we create an instance of the model, there are a lot of parameters you can choose to define and customize here, we will only define the `hidden_layer_sizes`. For this parameter you pass in a tuple consisting of the number of neurons you want at each layer, where the `nth` entry in the tuple represents the number of neurons in the `nth` layer of the MLP model. There are many ways to choose these numbers, but for simplicity we will choose 3 layers with the same number of neurons as there are features in our data set:

```
In [9]: # create a Multilayerperceptron classifier and call it mlp
mlp = MLPClassifier(hidden_layer_sizes = (30, 30, 30))
```

Now that the model has been made we can fit the training data to our model, remember that this data has already been processed and scaled:

```
In [10]: mlp.fit(X_train_transformed, y_train)
```

```
Out[10]: 

▼
  MLPClassifier
  MLPClassifier(hidden_layer_sizes=(30, 30, 30))


```

**Q:** What do you see in the output? What does it tell you?

There are 3 hidden layers with 30 neurons each. The number of neurons in a layer is dependent on the number of features in the data.

## Predictions and Evaluation

Now that we have a model it is time to use it to get predictions! We can do this simply with the `predict()` method off of our fitted model:

```
In [11]: y_pred = mlp.predict(X_test_transformed)
y_pred
```

```
Out[11]: array([0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
        1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
        0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
        1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1])
```

Now we can use SciKit-Learn's built in metrics such as a classification report and confusion matrix to evaluate how well our model performed:

```
In [12]: from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

```
[[57  4]
 [ 0 82]]
```

	precision	recall	f1-score	support
0	1.00	0.93	0.97	61
1	0.95	1.00	0.98	82
accuracy			0.97	143
macro avg	0.98	0.97	0.97	143
weighted avg	0.97	0.97	0.97	143

**Q:** what conclusion can you make from the confusion matrix?

I can conclude that my model has high precision, recall, and F1-score for the two outputs, False and True, and an accuracy of 97% to be able to make correct predictions for the negative and positive samples.

### Weights and biases

The downside however to using a Multi-Layer Preceptron model is how difficult it is to interpret the model itself. The weights and biases won't be easily interpretable in relation to which features are important to the model itself.

To extract the MLP weights and biases after training your model, you use its public attributes `coefs_` and `intercepts_`.

In [13]: *# Print the coefficient values and interpret it*

```
coefficients = mlp.coefs_[0]
coefficients
```

```
Out[13]: array([[ -0.1348932 , -0.26243641, -0.13492215, -0.20923182, -0.04909902,
         0.11550678, -0.08054107,  0.29168487,  0.14553945, -0.12810964,
        -0.1472266 ,  0.18486843, -0.12295266,  0.30723479, -0.20584349,
         0.30581277,  0.1480443 , -0.1741783 , -0.28915813,  0.02187519,
         0.26217966,  0.02806523, -0.24489932,  0.27378627,  0.17576296,
        -0.15748238,  0.22920037,  0.09730606,  0.23248364, -0.14839052],
        [ 0.20330617, -0.30123041, -0.12612949,  0.21585199, -0.18347869,
        -0.05822357,  0.09896193,  0.31184303, -0.22004983, -0.29374942,
         0.16367996,  0.23961324, -0.06863384,  0.02124192, -0.21554987,
        -0.19529145,  0.25073918,  0.25729685,  0.22818582, -0.16684243,
        -0.09770845,  0.02945036, -0.37682906, -0.01773573,  0.00567062,
        -0.24167704,  0.06469813, -0.09513673,  0.009529 ,  0.34533288],
        [-0.02966327, -0.23266791,  0.11121862,  0.11956834, -0.34718341,
        -0.38615165,  0.23787106, -0.05344889,  0.3161587 , -0.13118013,
         0.07776169, -0.29184155,  0.39052985,  0.02296943,  0.10667326,
         0.00150195, -0.10493366, -0.17930562, -0.334441 ,  0.09639755,
        -0.24291226,  0.07379263,  0.02572206, -0.21411073, -0.25649903,
         0.17897422,  0.00522721,  0.1529925 , -0.02137358,  0.26487154],
        [-0.35130624, -0.26532139,  0.1372096 ,  0.13133109,  0.03618637,
```

In [14]: `len(coefficients)`

Out[14]: 30

In [15]: *# Print the intercepts values and interpret it*

```
interprets = mlp.intercepts_[0]
interprets
```

```
Out[15]: array([ 0.00864307, -0.12179751,  0.00199165,  0.12632815, -0.19063782,
        -0.05828515,  0.06638915,  0.0709988 , -0.31433613, -0.1099543 ,
         0.07909943,  0.1116525 , -0.210572 , -0.14644128,  0.40757044,
         0.14390508, -0.25824413, -0.06137398, -0.04668632,  0.17117016,
        -0.15644084, -0.20505539,  0.07235976, -0.13391416,  0.15935218,
        -0.0383094 , -0.09495383,  0.40421205,  0.29538941,  0.38537121])
```

In [16]: `len(interprets)`

Out[16]: 30

**Q:** What do you understand from the two values?

Weight determine the influence input data has on the output, and can prevent overfitting of the learning model. Bias provide the information to efficiently propagate forward the learning model.

