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)](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_n
ames', '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 StandardScalar 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?

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([1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 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))
```

```
[[45  7]
 [ 0 91]]
              precision    recall  f1-score   support

           0       1.00      0.87      0.93        52
           1       0.93      1.00      0.96        91

    accuracy                           0.95       143
   macro avg       0.96      0.93      0.95       143
weighted avg       0.95      0.95      0.95       143
```

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

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([[-1.20962457e-01,  3.79080362e-02, -9.52223589e-02,
         2.06961461e-01, -2.67149682e-01,  2.17801254e-01,
        -7.80601794e-03, -8.58694275e-02,  2.57208210e-01,
        -3.88849110e-02, -3.17519280e-01, -4.11738920e-02,
        -1.12559062e-01, -2.15706127e-01,  1.88379964e-01,
        -2.00196008e-01, -2.71399961e-01, -1.23042706e-01,
         1.49592481e-01,  1.21038564e-01,  1.70979328e-01,
         5.12460564e-03,  7.15706031e-02, -4.74034648e-02,
        -2.16449477e-01, -3.00643740e-01, -1.70523389e-01,
         1.03457085e-01,  2.17544456e-01, -7.79958586e-02],
       [-2.95623488e-02, -5.91936886e-02,  1.45247923e-01,
        -4.62336011e-02, -1.13118125e-01, -6.89137241e-02,
         1.00788478e-01, -2.18879062e-01,  1.54537142e-01,
         6.00636732e-02,  4.24197601e-02, -2.97185364e-01,
        -2.79195058e-02,  9.63840191e-02, -2.46688812e-01,
        -1.80610344e-02, -5.42686319e-03,  2.23211874e-01,
        -2.43241330e-01, -3.57352931e-01,  2.01147450e-01,
        -1.54724154e-01, -2.25223374e-01, -3.42740825e-02,
         1.70069468e-03,  1.00573786e-01, -1.56248512e-01,
```

In [14]:
```
len(coefficients)
```

Out[14]: 30

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

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

Out[15]:
```
array([ 0.2601649 ,  0.19751712,  0.22924852, -0.04097408,  0.19407671,
       -0.09669076, -0.14411879,  0.3861132 , -0.07689121,  0.16068376,
       -0.19028552,  0.02001402, -0.06931704,  0.05647909,  0.06633307,
        0.21271881, -0.16759769, -0.18336963, -0.08017066, -0.11467018,
        0.33614298, -0.18520528,  0.19741961,  0.17152835,  0.36039939,
       -0.08611969,  0.14616534, -0.18519178, -0.04129569,  0.18278985])
```

In [16]:
```
len(interprets)
```

Out[16]: 30

Q: What do you understand from the two values?