

Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe30 - Neural Network Exercise 2

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 #2

This exercise is adapted from <https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/> (<https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/>).

Now you have successfully used SciKit Learn's MLP to work on the built-in Breast Cancer Data Set, let's try another one!

Download the wine dataset from UCI Machine learning repository (<http://archive.ics.uci.edu/ml/datasets/Wine/> (<http://archive.ics.uci.edu/ml/datasets/Wine/>)).
Import the dataset into a pandas dataframe

```
In [1]: import pandas as pd

data = pd.read_csv('../data_samples2/Wine.csv')
```

Check out the dataframe - what are the first few rows of data?

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

data.head()
```

Out[2]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_
0	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	

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

data.shape
```

Out[3]: (178, 14)

```
In [4]: # use describe to find out more about the data

data.describe()
```

Out[4]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000

Q: what can you say about this dataset?

Now, set up the data (x) and labels (y)

```
In [5]: x = data.drop(columns=['Customer_Segment'])  
y = data['Customer_Segment']
```

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 [6]: 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 [7]: # Import the StandardScaler library  
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
  
# Fit only to the training data  
scaler.fit(x_train)
```

```
Out[7]: 

|                  |
|------------------|
| ▼ StandardScaler |
| StandardScaler() |


```

```
In [8]: # Now apply the transformations to the data:  
x_train_transformed = scaler.transform(x_train)  
x_test_transformed = scaler.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 [9]: 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

```
In [10]: # create a Multilayerperceptron classifier and call it mlp
mlp = MLPClassifier(hidden_layer_sizes = (13, 13, 13), max_iter=500)
```

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 [11]: mlp.fit(x_train_transformed, y_train)
```

```
Out[11]: MLPClassifier
MLPClassifier(hidden_layer_sizes=(13, 13, 13), max_iter=500)
```

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

There are 3 hidden layers with 13 neurons each. The number of neurons in a layer is dependent on the number of features in the data.

There is a maximum number of iterations (epochs), that is 500, that the neural network will undergo during training. An epoch is a complete pass through the entire training dataset. More epochs allows the model to potentially learn more complex patterns, but also increases the training time.

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 [12]: y_pred = mlp.predict(x_test_transformed)
y_pred
```

```
Out[12]: array([2, 1, 3, 1, 2, 2, 3, 3, 3, 2, 2, 2, 3, 2, 1, 1, 2, 1, 1, 2, 2, 3,
                1, 2, 3, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 3, 2, 2, 1, 1, 1, 3, 1, 2,
                3], dtype=int64)
```

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 [13]: from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))

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

```
[[14  0  0]
 [ 0 21  0]
 [ 0  0 10]]
```

	precision	recall	f1-score	support
1	1.00	1.00	1.00	14
2	1.00	1.00	1.00	21
3	1.00	1.00	1.00	10
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

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 three outputs, the three different customer segments, and an accuracy of 100% to be able to make correct predictions for the determining the right customer.

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

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

```
Out[14]: array([[ 0.13583521, -0.5666878 ,  0.0755029 , -0.1803674 , -0.12638721,
                -0.16346479,  0.33947293,  0.18390313,  0.14161698,  0.12153496,
                -0.36565643, -0.18623026, -0.2203828 ],
                [ 0.27320248, -0.27895894,  0.09077072, -0.5449823 ,  0.2332957 ,
                -0.45973004, -0.16673408,  0.24269424, -0.37599694,  0.24820159,
                 0.19999198,  0.21787563, -0.34004824],
                [ 0.32082263, -0.46356226, -0.60535609, -0.14729081,  0.18424747,
                -0.28397209, -0.13709483,  0.26210316,  0.03914274, -0.16243488,
                 0.23227945,  0.20200026, -0.49474781],
                [ 0.30192308,  0.43235626,  0.59232481, -0.02834522, -0.07846221,
                -0.2467008 , -0.47082855,  0.22795147,  0.40084027, -0.30010313,
                 0.4296271 ,  0.31999184, -0.38976074],
                [ 0.16436839,  0.33639376, -0.38578635,  0.04308997, -0.2965029 ,
                -0.33204394, -0.30049134,  0.04909651, -0.11852367, -0.23336593,
                -0.44138096,  0.54494039,  0.03446113],
                [-0.43191593, -0.09137777, -0.41167078,  0.27382797,  0.20623915,
                -0.21167297, -0.14538682, -0.43277523, -0.38766075, -0.02203012,
                -0.15506468,  0.40362413,  0.10521812],
                [-0.378371 , -0.05793354,  0.2896879 , -0.04332337, -0.57222618,
                 0.07548307,  0.01464981, -0.57559432,  0.40697781, -0.10789724,
                 0.1173588 ,  0.37039909, -0.29424671],
                [ 0.09450932, -0.21878944,  0.09353017, -0.1592505 , -0.22540839,
                -0.45393415, -0.42052334, -0.2011724 ,  0.27109058,  0.3230203 ,
                 0.22798108, -0.13604959,  0.23876782],
                [ 0.21296496,  0.26590149, -0.02264956,  0.23493427,  0.21065526,
                -0.19992363,  0.37470774, -0.56544642, -0.34934474,  0.18484538,
                 0.0293261 ,  0.10537916, -0.10841344],
                [ 0.37180922, -0.03415664, -0.00147283, -0.22443263,  0.17186946,
                -0.24011376,  0.31318037, -0.14075015, -0.39282051, -0.04311824,
                -0.26117544,  0.32098737,  0.06995383],
                [ 0.33186894,  0.60014242, -0.05972911,  0.39524318, -0.43767657,
                 0.63091979,  0.58022404,  0.18900188,  0.39111019, -0.3419918 ,
                -0.1097246 ,  0.56515895,  0.50726346],
                [-0.10255448,  0.05861581,  0.59502812,  0.03434173, -0.55974156,
                -0.17699819,  0.09376463, -0.62303267, -0.12339809,  0.13134618,
                -0.51463217, -0.1711483 , -0.12567423],
                [ 0.40801857, -0.22503206, -0.08982238,  0.50682442,  0.38678885,
                -0.46860905,  0.47787164,  0.24268261,  0.2238461 , -0.341433 ,
                -0.2414029 , -0.07081258, -0.20900332]])
```

In [15]: `len(coefficients)`

Out[15]: 13

```
In [16]: # Print the intercepts values and interpret it
```

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

```
Out[16]: array([ 0.26013067,  0.25532309,  0.48426901, -0.17066397,  0.54998328,  
                -0.00451345, -0.16167469, -0.13611107, -0.14567238, -0.18996157,  
                -0.02364626,  0.16631914, -0.21821057])
```

```
In [17]: len(interprets)
```

```
Out[17]: 13
```

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.

Additional optional tasks...

select a few known supervised techniques and compare their performance. Use 10 fold cross validation

```
In [ ]:
```