Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe30 - Neural Network Exercise 2

Name: Ooi Caaron

IC Number: 990701-07-5837

Date :2/8/23

Introduction : Learning neural network

Conclusion :

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 [7]:

```
import pandas as pd
wine = pd.read_csv('Wine.csv')
```

In [8]:

```
wine
```

Out[8]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonfla
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	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	

178 rows × 14 columns

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

In [11]:

```
# find out the attributes in the dataset
wine.head()
```

Out[11]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflava
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 [10]:

```
# find out the total instances and number of features
wine.shape
```

Out[10]:

```
(178, 14)
```

In [9]:

```
# use describe to find out more about the data
wine.describe()
```

Out[9]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavan
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080

Q: what can you say about this dataset?

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

In [13]:

```
X = wine.drop('Customer_Segment',axis=1)
y = wine['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 [14]:

```
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 [15]:

```
# Import the StandardScalar library

from sklearn.preprocessing import StandardScaler
# Fit only to the training data

scaler = StandardScaler()
scaler.fit(X_train)
```

Out[15]:

```
StandardScaler()
```

In [16]:

```
# Now apply the transformations to the data:
StandardScaler(copy=True, with_mean=True, with_std=True)
X_train = scaler.transform(X_train)
X_test = 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 [17]:

```
from sklearn.neural_network import MLPClassifier
```

Out[17]:

```
MLPClassifier(hidden_layer_sizes=(13, 13, 13), max_iter=500)
```

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 [22]:

```
# 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 [ ]:

```
mlp.fit(X_train,y_train)
```

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 [20]:

```
predictions = mlp.predict(X_test)
```

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 [21]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[19  1  0]
 [ 0 10  0]
 [ 0  0 15]]
```

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 [ ]:

```
# Print the coefficient values and interpret it
len(mlp.coefs_)
```

In [ ]:

```
# Print the intercepts values and interpret it
len(mlp.intercepts_[0])
```

Q: What do you understand from the two values?

Additional optional tasks...

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

In [ ]: