nnet_MLP_wine

September 10, 2023

Can we detect fake wines from their chemical analysis? The data—available from the UCI data repository—contain the chemical analysis of three cultivars of Italian wine. We want to develop a model that can automatically class a wine sample into one of the three varieties from the values of its anlaysis.

For this, the datatset has 13 attributes—chemical compounds—measured on 178 samples. We will attempt to fit an MLP model for this classification problem.

We begin by loading the data and examining the first few lines.

[1]:
```
import pandas as pd
wine = pd.read_csv('wine_data.csv', names = ["Cultivar", "Alchol",␣
 ↪"Malic_Acid", "Ash", "Alcalinity_of_Ash", "Magnesium", "Total_phenols",␣
 ↪"Falvanoids", "Nonflavanoid_phenols", "Proanthocyanins", "Color_intensity",␣
 ↪"Hue", "OD280", "Proline"])
wine.head()
```

[1]:

	Cultivar	Alchol	Malic_Acid	Ash	Alcalinity_of_Ash	Magnesium	\
0	1	14.23	1.71	2.43	15.6	127	
1	1	13.20	1.78	2.14	11.2	100	
2	1	13.16	2.36	2.67	18.6	101	
3	1	14.37	1.95	2.50	16.8	113	
4	1	13.24	2.59	2.87	21.0	118	

	Total_phenols	Falvanoids	Nonflavanoid_phenols	Proanthocyanins	\
0	2.80	3.06	0.28	2.29	
1	2.65	2.76	0.26	1.28	
2	2.80	3.24	0.30	2.81	
3	3.85	3.49	0.24	2.18	
4	2.80	2.69	0.39	1.82	

	Color_intensity	Hue	OD280	Proline
0	5.64	1.04	3.92	1065
1	4.38	1.05	3.40	1050
2	5.68	1.03	3.17	1185
3	7.80	0.86	3.45	1480
4	4.32	1.04	2.93	735

As an initial step of the exploratory datta analysis (EDA), we compute the elementary statistics.

[4]:
```
wine.describe().transpose()
```

[4]:

	count	mean	std	min	25%	\
Cultivar	178.0	1.938202	0.775035	1.00	1.0000	
Alchol	178.0	13.000618	0.811827	11.03	12.3625	
Malic_Acid	178.0	2.336348	1.117146	0.74	1.6025	
Ash	178.0	2.366517	0.274344	1.36	2.2100	
Alcalinity_of_Ash	178.0	19.494944	3.339564	10.60	17.2000	
Magnesium	178.0	99.741573	14.282484	70.00	88.0000	
Total_phenols	178.0	2.295112	0.625851	0.98	1.7425	
Falvanoids	178.0	2.029270	0.998859	0.34	1.2050	
Nonflavanoid_phenols	178.0	0.361854	0.124453	0.13	0.2700	
Proanthocyanins	178.0	1.590899	0.572359	0.41	1.2500	
Color_intensity	178.0	5.058090	2.318286	1.28	3.2200	
Hue	178.0	0.957449	0.228572	0.48	0.7825	
OD280	178.0	2.611685	0.709990	1.27	1.9375	
Proline	178.0	746.893258	314.907474	278.00	500.5000	

	50%	75%	max
Cultivar	2.000	3.0000	3.00
Alchol	13.050	13.6775	14.83
Malic_Acid	1.865	3.0825	5.80
Ash	2.360	2.5575	3.23
Alcalinity_of_Ash	19.500	21.5000	30.00
Magnesium	98.000	107.0000	162.00
Total_phenols	2.355	2.8000	3.88
Falvanoids	2.135	2.8750	5.08
Nonflavanoid_phenols	0.340	0.4375	0.66
Proanthocyanins	1.555	1.9500	3.58
Color_intensity	4.690	6.2000	13.00
Hue	0.965	1.1200	1.71
OD280	2.780	3.1700	4.00
Proline	673.500	985.0000	1680.00

[6]:
```
wine.shape
```

[6]: (178, 14)

0.0.1 Data Preparation

We should perform a complete EDA, but since we have already decided to fit an MLP, we will skip this stage. The data preparation entails the following steps:

- First, place the data into a data matrix of explanatory variables plus the repsonse variable.
- Then, divide the data into a training set and a test set.
- Finally, normalize the data since they have varying magnitudes. For this, we use the class `StandardScaler` on the training data, which can then be applied to the test data in a `pipeline`. An alternative would be to use the function `scale` directly.

[2]:
```
X = wine.drop('Cultivar',axis=1)
y = wine['Cultivar']
X.head()
```

[2]:

	Alchol	Malic_Acid	Ash	Alcalinity_of_Ash	Magnesium	Total_phenols	\
0	14.23	1.71	2.43	15.6	127	2.80	
1	13.20	1.78	2.14	11.2	100	2.65	
2	13.16	2.36	2.67	18.6	101	2.80	
3	14.37	1.95	2.50	16.8	113	3.85	
4	13.24	2.59	2.87	21.0	118	2.80	

	Falvanoids	Nonflavanoid_phenols	Proanthocyanins	Color_intensity	Hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	

	OD280	Proline
0	3.92	1065
1	3.40	1050
2	3.17	1185
3	3.45	1480
4	2.93	735

[13]:
```
y.head()
```

[13]:
```
0    1
1    1
2    1
3    1
4    1
Name: Cultivar, dtype: int64
```

[3]:
```
# Split into a trainig set and a test set (by defaltt 0.75 / 0.25)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
# Perform the normalization on the training data
from sklearn.preprocessing import StandardScaler
normaliser = StandardScaler()
normaliser.fit(X_train)
X_train = normaliser.transform(X_train)
X_test  = normaliser.transform(X_test)
X_train[:4,1:6]
```

[3]:
```
array([[ 1.77523647e-01, -2.77539145e-03,  9.10454986e-02,
         1.43559675e+00, -1.02994771e+00],
```

```
       [ 1.26873497e+00,  3.41373149e-02,  3.79356244e-01,
        -8.51042387e-01,  2.42670606e-02],
       [-7.81686310e-01, -9.99418462e-01, -1.20635286e+00,
        -1.36467656e-01,  1.86453948e-01],
       [-1.27449143e+00, -3.73095873e+00, -2.61907551e+00,
        -8.51042387e-01, -4.94730979e-01]])
```

[19]:
```
X_test[:4,1:6]
```

[19]:
```
array([[-0.97650873, -1.71293591, -0.26379385, -0.52825887,  0.12081638],
       [ 3.10018413, -0.99556948,  0.47952357, -0.93952772,  0.54887276],
       [-0.70533182,  0.325895  , -1.00711128,  0.56845808,  1.66840483],
       [ 0.72286657,  1.23204207,  1.07417751, -0.18553482, -1.21274389]])
```

0.0.2 Train the MLP model

We use the Multi-Layer Perceptron classifier, `MLPClassifier`, from the library `neural_network`

[4]:
```
from sklearn.neural_network import MLPClassifier
```

We can now create an instance of the model.

Among the numerous possible parameters, we only define

- the number of hidden layers,
- the number od neurons in each hidden layer.

For this, we send a list whose n-th element is equal to the number of neurons in hidden layer n. Here we choose 3 layers, with 13 neurons each, and we limit the number of iterations to 500.

[5]:
```
mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)
```

Having defined the model, we can now fit the training data, already prepared and normalized above.

[6]:
```
mlp.fit(X_train,y_train)
```

[6]:
```
MLPClassifier(hidden_layer_sizes=(13, 13, 13), max_iter=500)
```

The output shows all the default values, as well as the architecture that we defined. All of these could be modified and tuned.

0.1 Predictions and Model Evaluation

With the fitted model, we can now use the method `predict()` to make the actual predictions on the test data and print out the confusion matrix.

[7]:
```
previsions = mlp.predict(X_test)
```

[8]:
```
# confusion table
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,previsions))
```

```
[[16  0  0]
 [ 1 15  0]
 [ 0  0 13]]
```

We observe 2 bad lassifications out of 45.

[9]:
```
print(classification_report(y_test,previsions))
```

```
              precision    recall  f1-score   support

           1       0.94      1.00      0.97        16
           2       1.00      0.94      0.97        16
           3       1.00      1.00      1.00        13

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

We have an excellent classification rate of 96%.

0.1.1 Conclusions

1. An MLP model with 3 hidden layers, havinf 13 neurons each, provides a classifier with an accuracy rate of 96%.
2. For a more reliable estimate, we should perform cross-validation.
3. Other supervised learning methods could be used here:

- k-nn
- SVM
- Bagging, etc.