alessandro.pomponio2

February 7, 2021

1 Alessandro Pomponio - alessandro.pomponio2@studio.unibo.it

[1]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪plot_confusion_matrix

# Variables
file_name = 'exam_2021_01_27.csv'
separator = ','
random_state = 42

# Directives
%matplotlib inline
np.random.seed(random_state)
```

1. Load the data from the file and show: the first few rows, the output of the .describe() function, the number of rows and columns (4pt)
2. Since the data contain nulls, eliminate the rows with nulls
3. Since one of the predicting attributes is ordinal, it must be converted into numeric, you can use the OrdinalEncoder
4. Split the data into train and test
5. Use two classification models of your choice (say: model 1 and model 2) execute the tasks below
6. Model 1: find and show the best hyperparameter setting with cross validation on the training set, optimise for the best accuracy
7. Model 1: show the accuracy of classification and the confusion matrix on the test set For the confusion matrix use plot_confusion_matrix normalized in order to show for each class the precision (read carefully the documentation)
8. Model 2: find and show the best hyperparameter setting with cross validation on the training set, optimise for the best accuracy

9. Model 2: show the accuracy of classification and the confusion matrix on the test set For the confusion matrix use plot_confusion_matrix normalized in order to show for each class the precision (read carefully the documentation)

1.1 1. Load the data from the file and show: the first few rows, the output of the .describe() function, the number of rows and columns (4pt)

[2]:
```
# Load the data from the file
df = pd.read_csv(file_name, sep = separator, header = None)
```

[3]:
```
# Show the first few rows
df.head()
```

[3]:

	0	1	2	3	4
0	5.1	3.5	a	NaN	0
1	4.9	3.0	a	NaN	0
2	NaN	3.2	a	NaN	0
3	4.6	NaN	a	0.2	0
4	5.0	3.6	a	0.2	0

[4]:
```
# Show the output of the describe() function
df.describe()
```

[4]:

	0	1	3	4
count	141.000000	140.000000	137.000000	150.000000
mean	5.897872	3.036429	1.290511	1.000000
std	0.820232	0.437654	0.733934	0.819232
min	4.300000	2.000000	0.100000	0.000000
25%	5.200000	2.800000	0.400000	0.000000
50%	5.800000	3.000000	1.400000	1.000000
75%	6.400000	3.300000	1.800000	2.000000
max	7.900000	4.400000	2.500000	2.000000

[5]:
```
# Show the number of rows and columns
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns in this dataset")
```

```
There are 150 rows and 5 columns in this dataset
```

1.2 2. Since the data contain nulls, eliminate the rows with nulls

[6]:
```
# We will use Pandas's `dropna` function to eliminate the rows with nulls
df = df.dropna()
```

1.3 3. Since one of the predicting attributes is ordinal, it must be converted into numeric, you can use the OrdinalEncoder

The ordinal attribute is ‘2’

[7]:
```python
ordinal_attribute = '2'
```

[8]:
```python
# Set the transformer data type
transf_dtype = np.int32

# Instantiate the encoder and perform `fit_transform`
encoder = OrdinalEncoder(dtype = transf_dtype)
transformed = encoder.fit_transform(df)
```

Since `fit_transform` returns an `ndarray`, we will turn it back into a dataframe for convenience

[9]:
```python
df = pd.DataFrame(transformed)
```

1.4 4. Split the data into train and test

I assume the column '4' is our target class.

[10]:
```python
target = 4
```

Before splitting the data into the training and test sets, we need to divide the feature matrix from the class.

[11]:
```python
X = df.drop(target, axis = 1)
y = df[target]
```

[12]:
```python
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state =␣
 ↪random_state)
```

1.5 5. Use two classification models of your choice (say: model 1 and model 2) execute the tasks below

We will use Decision Trees (model 1) and K-Nearest Neighbors (model 2)

1.6 6. Model 1: find and show the best hyperparameter setting with cross validation on the training set, optimise for the best accuracy

[13]:
```python
# Instantiate the DecisionTree Classifier
dt = DecisionTreeClassifier(random_state = random_state)

# Fit it to the training data
dt.fit(Xtrain, ytrain)

# Create the range of parameters to try during cross-validation
dt_depths = range(1, dt.get_depth() + 1)

# We will use GridSearchCV to perform cross-validation
# we need to create the parameter list in a specific way
# for it to work
dt_params = [{'max_depth': list(dt_depths), 'random_state': [random_state]}]
```

```
# Instantiate GridSearchCV
dt_gs = GridSearchCV(   dt,
                        dt_params,
                        cv=5,
                        scoring='accuracy',             # as required
                        return_train_score = False,
                        n_jobs = 2,
                    )

# Fit it to the training data
dt_gs.fit(Xtrain, ytrain)

# Print the best parameters found
print(f"The best parameter found for the Decision Tree was {dt_gs.
 ↪best_params_}")
```

```
The best parameter found for the Decision Tree was {'max_depth': 3,
'random_state': 42}
```

1.7 7. Model 1: show the accuracy of classification and the confusion matrix on the test set. For the confusion matrix use plot_confusion_matrix normalized in order to show for each class the precision (read carefully the documentation)

In order to show the correct results, we will instantiate our Decision Tree with the best parameters we found and fit it to the data.

[14]:
```
dt = DecisionTreeClassifier(max_depth = 3, random_state = random_state)
dt.fit(Xtrain, ytrain)
```

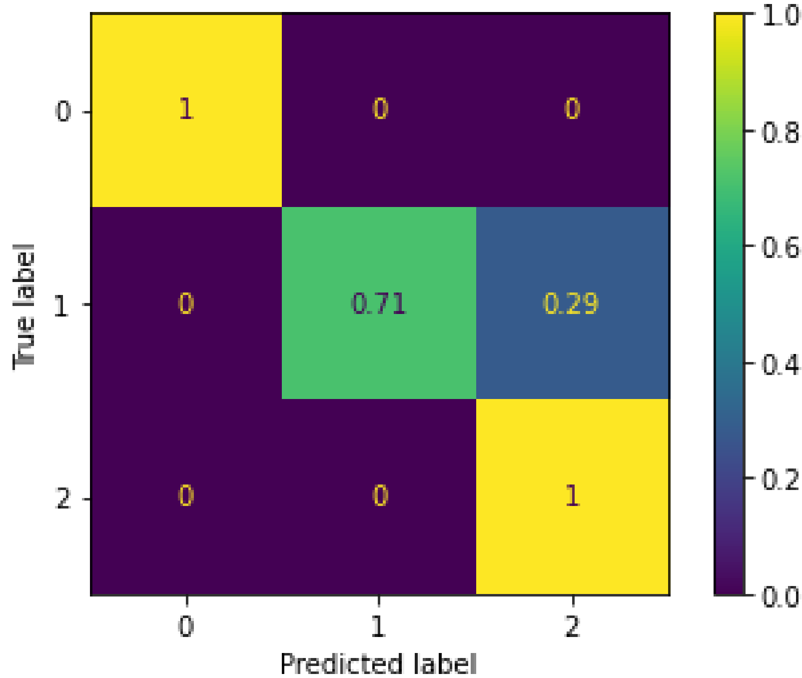
[14]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

[15]:
```
# Predict the test set in order to be able to compute the metrics later on
dt_ypred = dt.predict(Xtest)

# Accuracy score
dt_accuracy = accuracy_score(ytest, dt_ypred) * 100
print(f"The accuracy score for the Decision Tree with the optimized␣
 ↪hyperparameters was: {dt_accuracy:.2f}%")

# Confusion matrix normalized to show precision
print("The confusion matrix normalized to show precision is:")
plot_confusion_matrix(dt, Xtest, ytest, normalize = 'true');
```

```
The accuracy score for the Decision Tree with the optimized hyperparameters was:
87.10%
The confusion matrix normalized to show precision is:
```



1.8 8. Model 2: find and show the best hyperparameter setting with cross validation on the training set, optimise for the best accuracy

[16]:
```
# Instantiate the KNN Classifier
knn = KNeighborsClassifier()

# Create the range of parameters to try during cross-validation
knn_neighbors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# We will use GridSearchCV to perform cross-validation
# we need to create the parameter list in a specific way
# for it to work
knn_params = [{'n_neighbors': knn_neighbors}]

# Instantiate GridSearchCV
knn_gs = GridSearchCV(  knn,
                        knn_params,
                        cv=5,
                        scoring='accuracy',             # as required
                        return_train_score = False,
                        n_jobs = 2,
                )

# Fit it to the training data
```

```
knn_gs.fit(Xtrain, ytrain)

# Print the best parameters found
print(f"The best parameter found for the Decision Tree was {knn_gs.
 ↪best_params_}")
```

```
The best parameter found for the Decision Tree was {'n_neighbors': 4}
```

1.9 9. Model 2: show the accuracy of classification and the confusion matrix on the test set. For the confusion matrix use plot_confusion_matrix normalized in order to show for each class the precision (read carefully the documentation)

In order to show the correct results, we will instantiate our K-Nearest Neighbors classifier with the best parameters we found and fit it to the data.

[17]:
```
# Instantiate the KNN Classifier
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(Xtrain, ytrain)
```

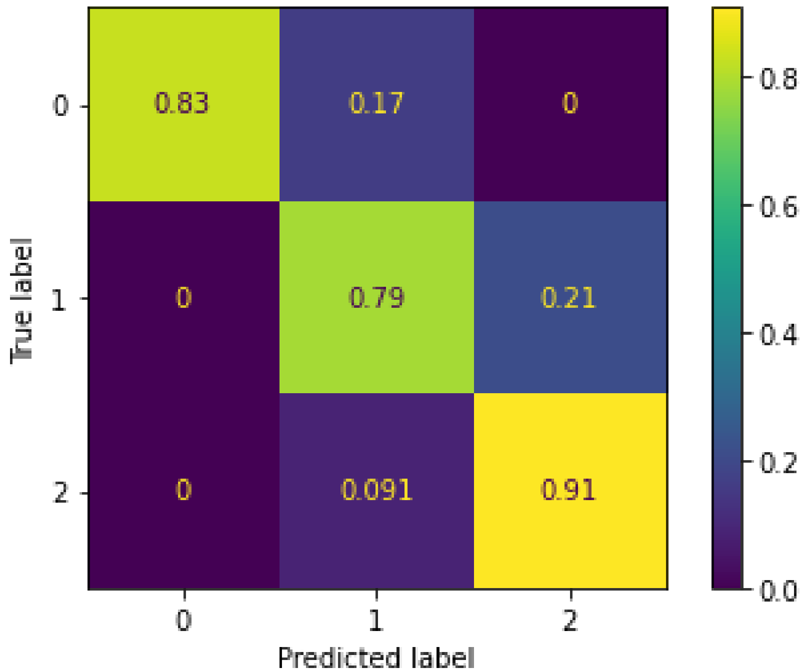
[17]:
```
KNeighborsClassifier(n_neighbors=4)
```

[18]:
```
# Predict the test set in order to be able to compute the metrics later on
knn_ypred = knn.predict(Xtest)

# Accuracy score
knn_accuracy = accuracy_score(ytest, knn_ypred) * 100
print(f"The accuracy score for the K-Nearest Neighbors classifier with the␣
 ↪optimized hyperparameters was: {knn_accuracy:.2f}%")

# Confusion matrix normalized to show precision
print("The confusion matrix normalized to show precision is:")
plot_confusion_matrix(knn, Xtest, ytest, normalize = 'true');
```

```
The accuracy score for the K-Nearest Neighbors classifier with the optimized
hyperparameters was: 83.87%
The confusion matrix normalized to show precision is:
```


0.83
0.17
0
0
0.79
0.21
0
0.091
0.91
True label
Predicted label
0.8
0.6
0.4
0.2
0.0