Pythonkurs - 03 - Mandag - Scikit Learn - Iris

December 6, 2024

0.1 Load and inspect the dataset

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
from sklearn import set_config
set_config(display="diagram")
```

[3]:
```
# Load scikit learn and load the dataset
from sklearn.datasets import load_iris # conda install scikit-learn, optional:␣
↪conda install scikit-learn-intelex
iris_dataset = load_iris()
```

[4]:
```
type(iris_dataset)
```

[4]:
```
sklearn.utils._bunch.Bunch
```

[5]:
```
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))
```

```
Keys of iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

[6]:
```
print("First five columns of data:\n{}".format(iris_dataset['data'][:5]))
print("\n")
print("Targets:\n{}".format(iris_dataset['target'][:]))
print("\n")
print("Target names:\n{}".format(iris_dataset['target_names']))
print("\n")
print("Feature names:\n{}".format(iris_dataset['feature_names']))
print("\n")
print("Dataset location:\n{}".format(iris_dataset['filename']))
```

```
First five columns of data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

```
Targets:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]


Target names:
['setosa' 'versicolor' 'virginica']


Feature names:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']


Dataset location:
iris.csv
```

[7]:
```
print(iris_dataset['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
```

```
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).
  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
    Structure and Classification Rule for Recognition in Partially Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more …
```

0.2 Prepare data for training the model

[8]:
```
# We need to create both a training set AND a testing set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
```

```
        iris_dataset['data'],
        iris_dataset['target'],
        test_size=0.25, # represent the proportion of the dataset to include in␣
 ↪the test split.
        random_state=42 # an int for reproducible output across multiple␣
 ↪function calls.
)
```

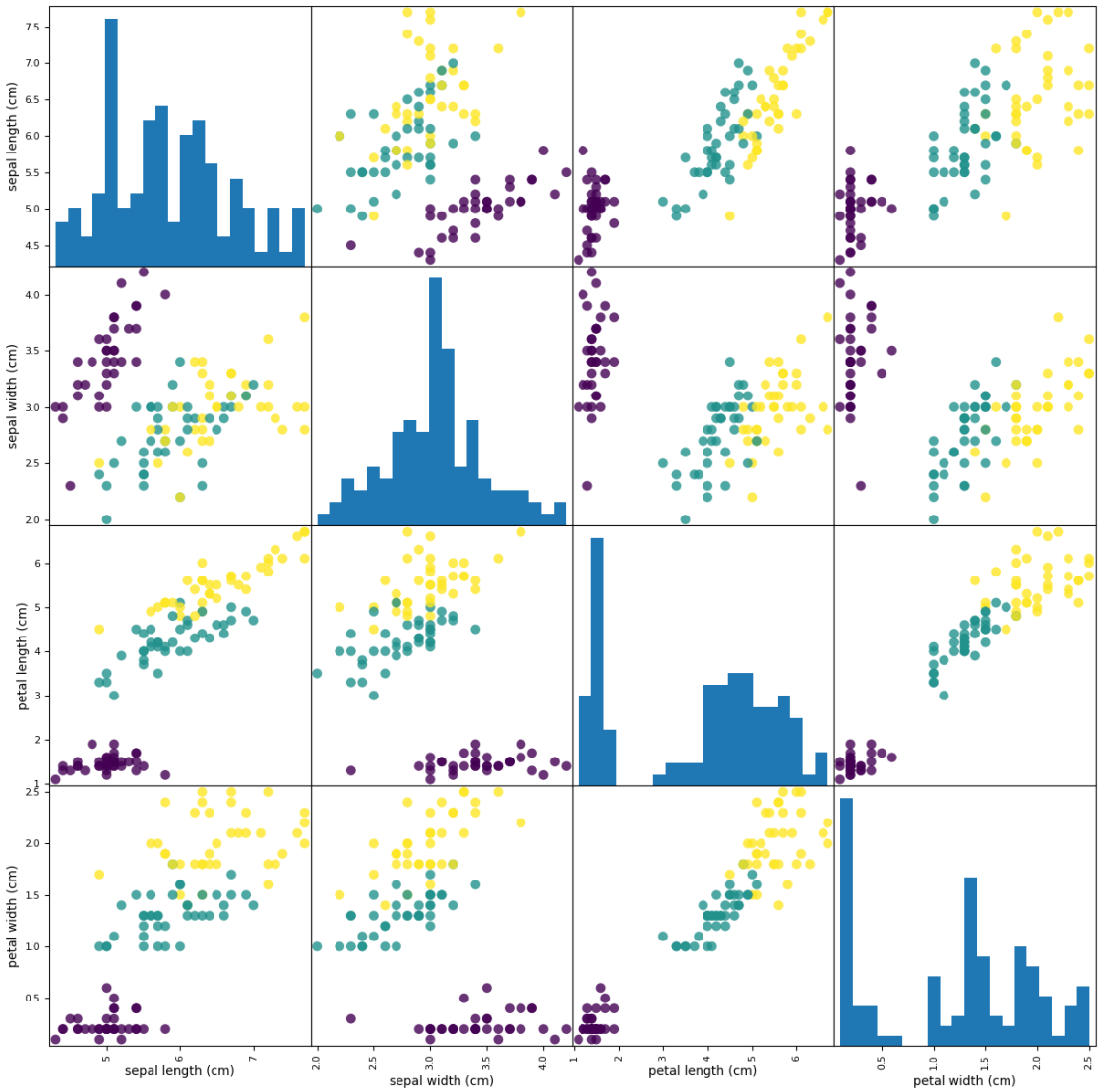
[9]:
```
print("X_train type: ", type(X_train))
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train type:  <class 'numpy.ndarray'>
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

[10]:
```
# conda install matplotlib
import pandas as pd
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

print(iris_dataframe)
print(y_train)

grr = pd.plotting.scatter_matrix(
    iris_dataframe,
    c=y_train, # color by category.
    figsize=(15, 15),
    marker='o',
    hist_kwds={'bins': 20}, # keywords to hist function.
    s=60, # marker size in points**2 (typographic points are 1/72 in. Default␣
 ↪is rcParams['lines.markersize'] ** 2.
    alpha=.8
)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.0	3.6	1.4	0.2
1	5.2	4.1	1.5	0.1
2	5.8	2.7	5.1	1.9
3	6.0	3.4	4.5	1.6
4	6.7	3.1	4.7	1.5
..	…	…	…	…
107	6.1	2.8	4.0	1.3
108	4.9	2.5	4.5	1.7
109	5.8	4.0	1.2	0.2

```
110                5.8                2.6                4.0                1.2
111                7.1                3.0                5.9                2.1

[112 rows x 4 columns]
[0 0 2 1 1 0 0 1 2 2 1 2 1 2 1 0 2 1 0 0 0 1 2 0 0 0 1 0 1 2 0 1 2 0 2 2 1
 1 2 1 0 1 2 0 0 1 1 0 2 0 0 1 1 2 1 2 2 1 0 0 2 2 0 0 0 1 2 0 2 2 0 1 1 2
 1 2 0 2 1 2 1 1 1 0 1 1 0 1 2 2 0 1 2 2 0 2 0 1 2 2 1 2 1 1 2 2 0 1 2 0 1
 2]
```

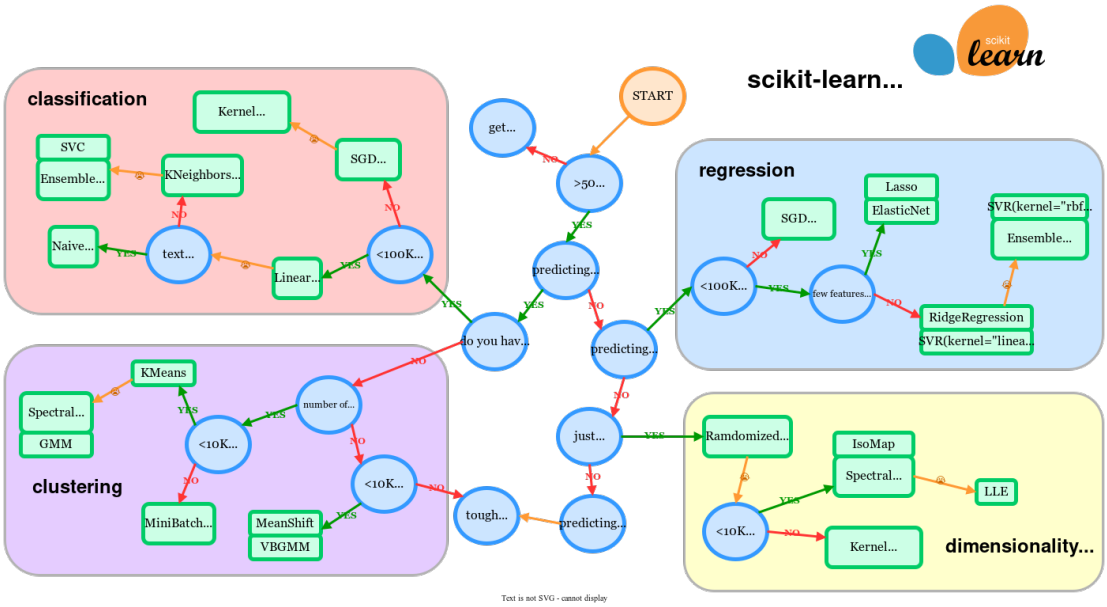


0.3 Build the model

[11]:
```
# Import one of many classification algorithms.
from sklearn.neighbors import KNeighborsClassifier

n_neighbors=1

# This is a object containing the algorithm that build the model.
knn = KNeighborsClassifier(n_neighbors=1)
```

[12]:
```
knn.fit(X_train[:,2:], y_train) # Slicing notation for numpyarray. We only want␣
↪the two last columns / dimensions.
```

[12]:
```
KNeighborsClassifier(n_neighbors=1)
```



https://scikit-learn.org/stable/machine_learning_map.html

0.4 Visualize our model

[13]:
```
import numpy as np

# Numpy arange()
print(np.arange(1, 2, 0.2))


# Numpy ravel()
a = np.array([[1, 2], [3, 4]])
b = np.ravel(a)
print(b)
```

```
# Numpy columnstacking: Translates slice objects to concatenation along the␣
 ↪second axis.
stack_a = np.c_[np.array([1,2,3]), np.array([4,5,6])]
stack_b = np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])]
print(stack_a)
print(stack_b)
```

```
[1.  1.2 1.4 1.6 1.8]
[1 2 3 4]
[[1 4]
 [2 5]
 [3 6]]
[[1 2 3 0 0 4 5 6]]
```

[14]:
```
# Lets visualize our trained model. The result will show the decision␣
 ↪boundaries of the model.
# Plot our training data set on top of a coloured domain in which each color␣
 ↪represents a category in our model.
import numpy as np

x_min,x_max = X_train[:,2].min() - 1, X_train[:,2].max()+ 1 # Get boundaries X.
y_min,y_max = X_train[:,3].min() - 1, X_train[:,3].max()+ 1 # Get boundaries Y.

print("Our boundaries")
print(x_min,x_max)
print(y_min,y_max)

h=0.02 # Spacing between values.
xx,yy = np.meshgrid(np.arange(x_min,x_max,h),np.arange(y_min,y_max,h)) # np.
 ↪arange() return evenly spaced values within a given interval.

print("\nResult of meshgrid based on our boundaries")
print(xx)
print(yy)
```

```
Our boundaries
0.10000000000000009 7.7
-0.9 3.5

Result of meshgrid based on our boundaries
[[0.1  0.12 0.14 … 7.64 7.66 7.68]
 [0.1  0.12 0.14 … 7.64 7.66 7.68]
 [0.1  0.12 0.14 … 7.64 7.66 7.68]
 …
 [0.1  0.12 0.14 … 7.64 7.66 7.68]
 [0.1  0.12 0.14 … 7.64 7.66 7.68]
 [0.1  0.12 0.14 … 7.64 7.66 7.68]]
[[-0.9  -0.9  -0.9  … -0.9  -0.9  -0.9 ]
```

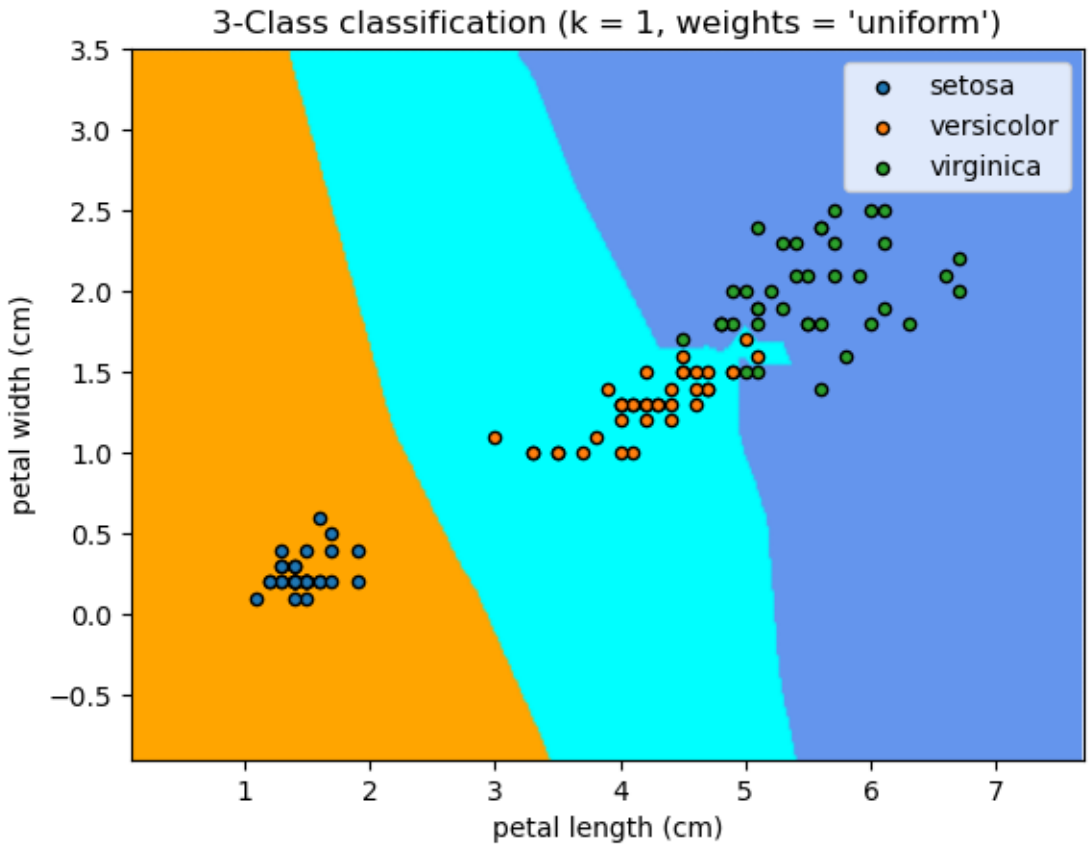
```
 [-0.88 -0.88 -0.88 … -0.88 -0.88 -0.88]
 [-0.86 -0.86 -0.86 … -0.86 -0.86 -0.86]
 …
 [ 3.44  3.44  3.44 …  3.44  3.44  3.44]
 [ 3.46  3.46  3.46 …  3.46  3.46  3.46]
 [ 3.48  3.48  3.48 …  3.48  3.48  3.48]]
```

[15]:
```
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()]) # Predict the class labels for␣
 ↪the provided data.
Z = Z.reshape(xx.shape) # Gives a new shape to an array without changing its␣
 ↪data.

print(type(Z))
print(Z.shape)
print(Z)
```

```
<class 'numpy.ndarray'>
(220, 380)
[[0 0 0 … 2 2 2]
 [0 0 0 … 2 2 2]
 [0 0 0 … 2 2 2]
 …
 [0 0 0 … 2 2 2]
 [0 0 0 … 2 2 2]
 [0 0 0 … 2 2 2]]
```

[16]:
```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt

# Define a suitable colorscheme.
cmap_bold = ListedColormap(['darkorange', 'blue', 'darkblue'])
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])

# Plot setup.
fig = plt.figure()
ax1 = fig.add_subplot(111) # (nrows, ncols, index)
ax1.pcolormesh(xx, yy, Z, cmap=cmap_light, shading='gouraud') # Create a␣
 ↪pseudocolor plot with a non-regular rectangular grid.

# Plot our observations.
for target in iris_dataset.target_names:
    index=np.where(iris_dataset.target_names==target)[0][0]
    ax1.scatter(
        X_train[:,2][y_train==index],
        X_train[:,3][y_train==index],
        cmap=cmap_bold,
        edgecolor='k',
        s=20,
```

```
        label=target
    )

# Some housekeeping on the plot.
ax1.set_xlim(x_min,x_max)
ax1.set_ylim(y_min,y_max)
ax1.legend()
ax1.set_xlabel("petal length (cm)")
ax1.set_ylabel("petal width (cm)")
ax1.set_title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, 'uniform'))
plt.show()
```



0.5 Use the model

[17]:
```
# Making a prediction.
#new_data = np.array([[4,3.5,1.2,0.5]]) # sepal length (cm),  sepal width (cm),␣
  ↪ petal length (cm),  petal width (cm)
newer_data = np.array([[5.5, 2.9, 4.0, 1.4]])
```

```
prediction = knn.predict(newer_data[:,2:]) #  We trained only with the two last␣
 ↪columns / dimensions.
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".
 ↪format(iris_dataset['target_names'][prediction]))
```

```
Prediction: [1]
Predicted target name: ['versicolor']
```

0.6 Validate the model

[18]:
```
# Validate the model using the testdata we prepared earlier.
y_pred = knn.predict(X_test[:,2:]) # We trained only with the two last columns /
 ↪ dimensions.
print("Test set predictions:\n {}".format(y_pred))
```

```
Test set predictions:
 [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0]
```

[19]:
```
print("Test set score: {:.2f}".format(knn.score(X_test[:,2:], y_test)))
```

```
Test set score: 1.00
```

0.7 Oppsummering

k-Nearest Neighbors is a simple classification algorithm in which predictions a new data point to the closest data points in the training dataset.

It is not necessary to use all the features in our training dataset. We can use different combinations to try to achive better results.