Working with Datasets in Scikit-learn

Neba Nfonsang
University of Denver

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
```

The datasets module in sklearn contains several datasets and other functionalities. To inspect the content of the dataset module, you can call dir() on the module or run ?module_name.*

```
dir(datasets)
# alternatively run
?datasets.*
```

For example, the following methods are used to load different datasets in sklearn:

datasets.load_boston
datasets.load_breast_cancer
datasets.load_diabetes
datasets.load_digits
datasets.load_files
datasets.load_iris
datasets.load_linnerud
datasets.load_mlcomp
datasets.load_sample_image
datasets.load_sample_images
datasets.load_svmlight_file
datasets.load_svmlight_files
datasets.load_wine

In [2]:

```
# load the boston dataset
boston = datasets.load_boston()
type(boston)
```

Out[2]:

```
sklearn.utils.Bunch
```

The Bunch Type

The loaded dataset is not a NumPy array. It is a Bunch object which is basically Python dictionary with keys as "data", target, "feature_names", "DESCR".

- The "data" key stores the input feature values as a NumPy array
- the "target key stores the output values as a vector
- the feature_names key stores the names of the features as a NumPy array
- the "DESCR" key stores the description of the dataset as a string

Input Features and Output

- In sklearn, the input features and output features are created as separate objects and are expected to be NumPy arrays
- The input features are assigned to the variable, X, which is a matrix whose rows represent the observations and columns represent represent the features.
- The output values are assigned to the variable, y, which is a vector whose length is the number of observations.

In [3]:

```
# input features
X = boston.data
X
```

Out[3]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300
e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800
e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800
e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000
e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000
e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000
e+01, 3.9690e+02,
        7.8800e+00]])
```

In [4]:

```
# shape of input features
X.shape
```

Out[4]:

```
(506, 13)
```

In [5]:

```
# view the names of the features
boston.feature_names
```

Out[5]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AG
E', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [6]:

```
# output
y = boston.target
# view just the first 20 elements of the vector
y[0:20]
```

Out[6]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1,
16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2,
18.2])
```

In [7]:

```
# shape of the output
y.shape
```

Out[7]:

```
(506,)
```

Description of the Dataset

In [8]:

```
print(boston.DESCR)
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predi
ctive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned
for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business ac
res per town
        - CHAS     Charles River dummy variable (= 1 if
tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts p
er 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units b
uilt prior to 1940
        - DIS      weighted distances to five Boston em
ployment centres
        - RAD      index of accessibility to radial hig
hways
        - TAX      full-value property-tax rate per $1
0,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the pr
oportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes
in $1000's
```

```
    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing (http://
archive.ics.uci.edu/ml/datasets/Housing)


This dataset was taken from the StatLib library which i
s maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfe
ld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Econo
mics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch,
 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are u
sed in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machi
ne learning papers that address regression
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Id
entifying Influential Data and Sources of Collinearit
y', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Mo
del-Based Learning. In Proceedings on the Tenth Interna
tional Conference of Machine Learning, 236-243, Univers
ity of Massachusetts, Amherst. Morgan Kaufmann.
   - many more! (see http://archive.ics.uci.edu/ml/data
sets/Housing) (http://archive.ics.uci.edu/ml/datasets/H
ousing))
```

NB: Use .target_names on the bunch object to print unique values in case of a categorical target variable

Creating Artificial Datasets

The datasets module provides a suite of functions used to create datasets for various types of problems including classification and regression problems.

- The make_blob function can be used to generate blobs of points with Gausian distribution, for clustering.
- The make_regression() can be used to generate data for a regression problem

In [9]:

```
np.random.seed(2020)
# the output are created with a regression model
# that takes the inputs and some random noise
X, y = datasets.make_regression(n_samples=1000, n_features=6, noise=
```

In [10]:

```
# view five rows of the input data
X[0:5]
```

Out[10]:

```
array([[ 4.07987640e-01, -7.36910500e-01,  3.58430940e-
01,
         5.25254892e-02, -4.53101681e-01, -3.93192496e-
01],
       [ 2.51482827e-01,  1.14580104e-01, -9.01729688e-
01,
        -7.02847575e-01, -1.10766715e-01, -1.24786288e+
00],
       [ 6.14419218e-01,  1.01033742e+00,  3.76799909e-
01,
         8.43624298e-01, -8.75778854e-01,  9.24659769e-
02],
       [-4.87580588e-01,  1.01682675e+00, -7.74962116e-
01,
         1.70139334e-01,  5.18023076e-01,  6.82801726e-
01],
       [ 4.54732557e-01, -5.62546144e-02, -6.34296858e-
01,
         1.07347952e-03, -1.23792993e+00, -1.24673112e+
00]])
```

In [11]:

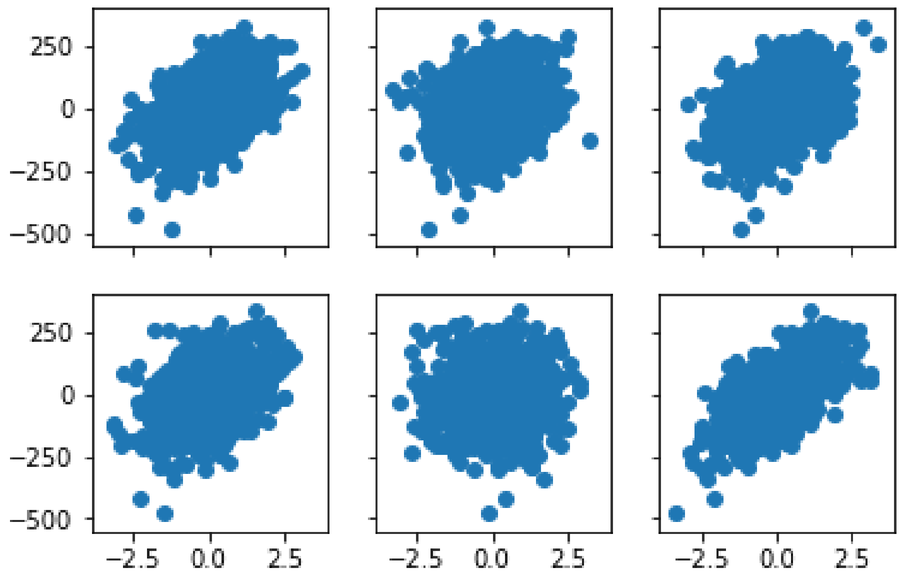
```
# view five data points of the output
y[0:5]
```

Out[11]:

```
array([ -18.09813332, -145.62747767,  106.20402367,   3
4.73906613,
       -113.26509825])
```

In [12]:

```
# Loop through each feature and plot it with the output.
fig, ax = plt.subplots(2, 3, sharex=True, sharey=True)
col = 0
for i in range(2):
    for j in range(3):
        ax[i, j].scatter(X[:, col], y)
        col = col + 1
```



In [13]:

```
# Create data for clustering
```

In [14]:

```
X, y = datasets.make_blobs(n_samples=1000, n_features=2, centers=3)
```

In [15]:

```
X
```

Out[15]:

```
array([[ -5.68105897,   7.0963262 ],
       [ -9.25581243,  -4.93680392],
       [ -7.03180299,   4.39385459],
       ...,
       [-10.1402933 ,  -5.96037749],
       [  6.37546226,  -1.41595469],
       [ -9.23958739,  -3.92505811]])
```

In [16]:

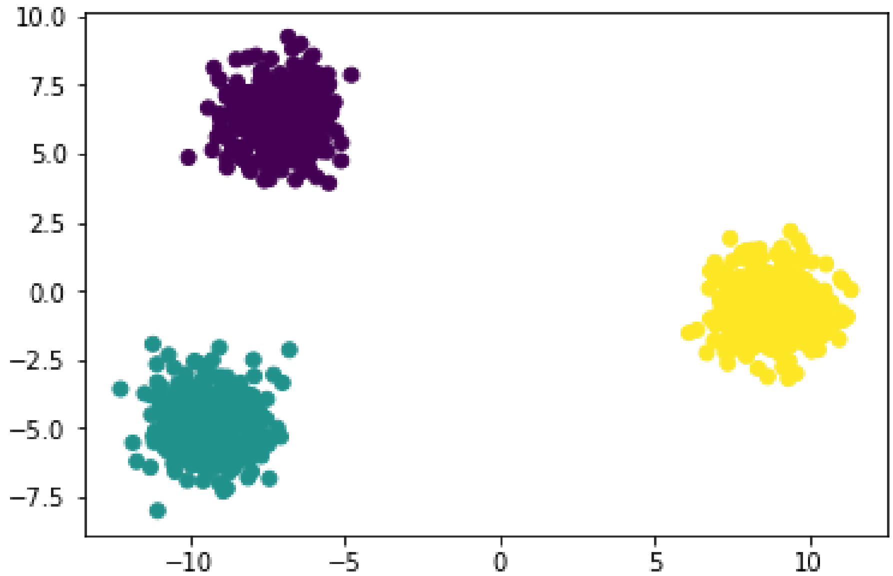
```
y[0:50]
```

Out[16]:

```
array([0, 1, 0, 0, 2, 2, 0, 0, 1, 0, 0, 2, 1, 1, 2, 1,
2, 2, 2, 1, 1, 2,
       0, 2, 2, 0, 1, 2, 2, 2, 0, 0, 0, 2, 1, 1, 2, 2,
2, 2, 0, 2, 1, 2,
       2, 0, 2, 0, 0, 1])
```

In [17]:

```
# create scatter and color points by output label
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



Generate the Regression Data from Scratch

In [18]:

```
n_samples = 1000
n_features = 6
np.random.seed(2020)
X = np.random.randn(n_samples, n_features)
X = np.round(X, 2)
X[0:10]
```

Out[18]:

```
array([[-1.77,  0.08, -1.13, -0.65, -0.89, -1.27],
       [-0.06,  0.06,  0.41, -0.57, -0.8 ,  1.31],
       [ 1.27, -1.21,  0.31, -1.44, -0.37, -0.77],
       [ 0.39,  0.06,  2.09,  0.04, -0.05, -0.51],
       [-0.08, -1.22, -1.41, -1.49,  0.38,  0.94],
       [ 1.77,  0.88,  0.33, -0.31,  1.24, -0.22],
       [ 0.16,  0.1 ,  0.83,  2.05, -0.32, -1.31],
       [-1.75,  0.1 , -1.36,  0.48, -0.21, -0.09],
       [ 0.7 ,  0.1 ,  0.62,  0.95,  2.04, -0.48],
       [ 0.21,  1.64, -0.49, -0.02,  0.47,  0.28]])
```

In [19]:

```
np.random.seed(2020)
n_parameters = n_features + 1
param_values = np.random.random(n_parameters)
param_values
```

Out[19]:

```
array([0.98627683, 0.87339195, 0.50974552, 0.27183571,
0.33691873,
       0.21695427, 0.27647714])
```

In [20]:

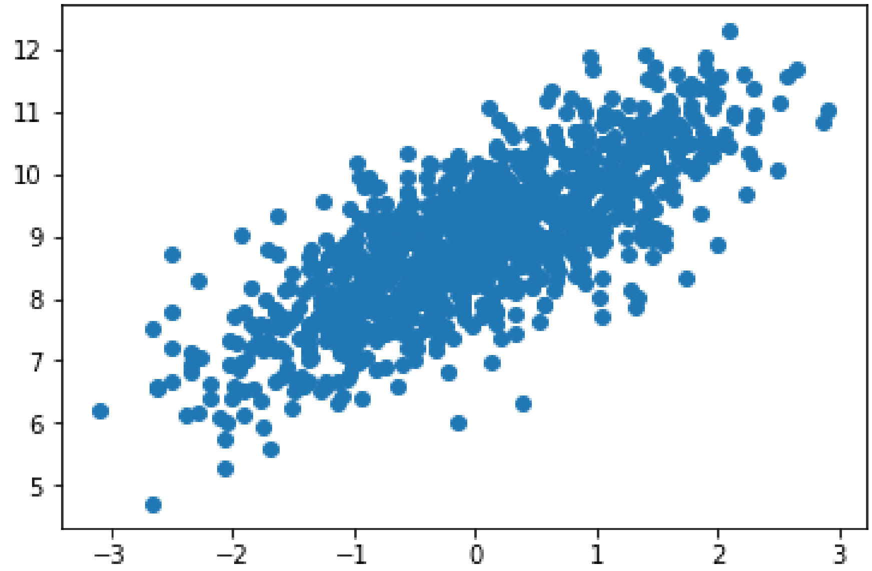
```
e = 0.5 # random noise
# y = linear function + random noise
y = (param_values[0] + X*param_values[1:] + e).sum(axis=1)
y.shape
```

Out[20]:

```
(1000,)
```

In [21]:

```
# plot the first feature and the output
plt.scatter(X[:,0], y)
plt.show()
```



Check More Datasets to Load

```
# run this code to check on available datasets
datasets.load_*?
```

the following datasets would be displayed:

datasets.load_boston
datasets.load_breast_cancer
datasets.load_diabetes
datasets.load_digits
datasets.load_files
datasets.load_iris
datasets.load_linnerud
datasets.load_mlcomp
datasets.load_sample_image
datasets.load_sample_images
datasets.load_svmlight_file
datasets.load_svmlight_files
datasets.load_wine

In [22]:

```
# load the iris dataset
iris = datasets.load_iris()
```

In [23]:

```
# display 10 rows of the input feature data
iris.data[0:10]
```

Out[23]:

```
array([[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],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [24]:

```
# display 100 values in the target output
iris.target[0:100]
```

Out[24]:

```
array([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, 1, 1, 1])
```

In [25]:

```
# display unique target output values
np.unique(iris.target)
```

Out[25]:

```
array([0, 1, 2])
```

In [26]:

```
# display the feature names
iris.feature_names
```

Out[26]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [27]:

```
# display the values in the target output
iris.target_names
```

Out[27]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U1
0')
```

In [28]:

```
# stack the input features and target output
# and display the first 10 rows
np.column_stack((iris.data, iris.target))[0:10]
```

Out[28]:

```
array([[5.1, 3.5, 1.4, 0.2, 0. ],
       [4.9, 3. , 1.4, 0.2, 0. ],
       [4.7, 3.2, 1.3, 0.2, 0. ],
       [4.6, 3.1, 1.5, 0.2, 0. ],
       [5. , 3.6, 1.4, 0.2, 0. ],
       [5.4, 3.9, 1.7, 0.4, 0. ],
       [4.6, 3.4, 1.4, 0.3, 0. ],
       [5. , 3.4, 1.5, 0.2, 0. ],
       [4.4, 2.9, 1.4, 0.2, 0. ],
       [4.9, 3.1, 1.5, 0.1, 0. ]])
```