

# Introduction à scikit-learn

Thèmes abordés : jeux de données, classification, regression, réduction de dimension, clustering

Davy Ouedraogo (Dpt. Informatique, Fac. Sciences, Université de Sherbrooke)

## 1 - Importation des bibliothèques

Nous importons les bibliothèques et modules nécessaires.

```
In [ ]: import sklearn
import numpy as np
import scipy as sp
import pandas as pd
%matplotlib notebook
import matplotlib.pyplot as plt
import seaborn as sns
```

## 2 - Chargement de données

Nous chargerons et analyserons superficiellement trois jeux de données du module ***sklearn.datasets***. Pour nos analyses de regression, nous choisirons le jeu de données ***load\_diabetes*** du module ***sklearn.datasets***.

### Module 'datasets' de scikit-learn

```
<code>
    from sklearn.datasets import dataset_class
    dataset_name = dataset_class()
    X = dataset_name['data']
    y = dataset_name['target']
</code>
```

### Besoins

- + Charger des jeux de données réels prédéfinis pour tester des fonctionnalités (Toy Example);
- + Générer des jeux de données simulés;
- + Charger des jeux de données réels non-prédéfinis.

## 2-1 iris dataset

```
In [ ]: from sklearn.datasets import load_iris
```

```
# Instancier un objet  
iris_dataset = load_iris()  
print("Keys of dataset: ", iris_dataset.keys())
```

Keys of dataset: dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

```
In [ ]: # Afficher le format des données de classification  
print("Data shape: ", iris_dataset['data'].shape)
```

Data shape: (150, 4)

```
In [ ]: # Afficher le format des données cibles  
print("Target shape: ", iris_dataset['target'].shape)
```

Target shape: (150,)

```
In [ ]: # Afficher les noms des attributs  
print("Feature names: ", iris_dataset['feature_names'])
```

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

```
In [ ]: # Afficher le type d'une valeur  
print("Type of first attribute: ", type(iris_dataset['data'][0,0]))
```

Type of first attribute: <class 'numpy.float64'>

```
In [ ]: # Afficher les valeurs d'un attribut  
print("20 first values of first attribute: ", iris_dataset['data'][:20,0])
```

20 first values of first attribute: [5.1 4.9 4.7 4.6 5. 5.4 4.6 5. 4.4 4.9 5.4 4.8 4.8 4.3 5.8  
5.7 5.4 5.1  
5.7 5.1]

```
In [ ]: # Afficher le type des valeurs cibles  
print("Type of target: ", type(iris_dataset['target'][0]))
```

Type of target: <class 'numpy.int32'>

```
In [ ]: # Afficher la description du jeu de données  
print("Description of dataset: ", iris_dataset['DESCR'])
```

Description of dataset: .. \_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.

.. topic:: 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 ...

## 2-2 breast\_cancer dataset (Classification binomiale ou binaire)

```
In [ ]: # Importer la classe et instancier un objet
from sklearn.datasets import load_breast_cancer
breast_cancer_dataset = load_breast_cancer()
```

```
In [ ]: # Afficher le format des données
print("Data shape: ", breast_cancer_dataset['data'].shape, "Target shape: ", breast_cancer_dataset['target'].shape)
```

Data shape: (569, 30) Target shape: (569,)

```
In [ ]: # Afficher la description du jeu de données
print("Description of dataset: ", breast_cancer_dataset['DESCR'][:1060])
```

Description of dataset: .. \_breast\_cancer\_dataset:

Breast cancer wisconsin (diagnostic) dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter<sup>2</sup> / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three worst/largest values) of these features were computed for each image, resulting in 30 features. For instance, field 0 is Mean Radius, field 10 is Radius SE, field 20 is Worst Radius.

## 2-3 wine dataset (multi-classification)

```
In [ ]: # Importer la classe, instancier un objet, et afficher le format des données
from sklearn.datasets import load_wine
wine_dataset = load_wine()
print("Data shape: ", wine_dataset['data'].shape, "Target shape: ", wine_dataset['target'].shape)
```

Data shape: (178, 13) Target shape: (178,)

```
In [ ]: # Afficher la description des données
print("Description of dataset: ", wine_dataset['DESCR'])
```

Description of dataset: .. \_wine\_dataset:

Wine recognition dataset

-----

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 178

:Number of Attributes: 13 numeric, predictive attributes and the class

:Attribute Information:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

- class:

- class\_0
- class\_1
- class\_2

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None

:Class Distribution: class\_0 (59), class\_1 (71), class\_2 (48)

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.

<https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

The data is the results of a chemical analysis of wines grown in the same

region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -  
An Extendible Package for Data Exploration, Classification and Correlation.  
Institute of Pharmaceutical and Food Analysis and Technologies,  
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository  
[<https://archive.ics.uci.edu/ml>]. Irvine, CA: University of California,  
School of Information and Computer Science.

.. topic:: References

(1) S. Aeberhard, D. Coomans and O. de Vel,  
Comparison of Classifiers in High Dimensional Settings,  
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Technometrics).

The data was used with many others for comparing various  
classifiers. The classes are separable, though only RDA  
has achieved 100% correct classification.  
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))  
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,  
"THE CLASSIFICATION PERFORMANCE OF RDA"  
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Journal of Chemometrics).

---

## 3 - Choix du jeu de données : Jeu de données 'diabetes' pour la regression

Lien: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)

[learn.org/stable/modules/generated/sklearn.datasets.load\\_diabetes.html#sklearn.datasets.load\\_diabetes](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)

---

```
In [ ]: # Importer la classe, instancier un objet, et afficher le format des données
from sklearn.datasets import load_diabetes
diabetes_dataset = load_diabetes()
print("Data shape: ", diabetes_dataset['data'].shape, "Target shape: ", diabetes_dataset['target']
```

Data shape: (442, 10) Target shape: (442,)

```
In [ ]: # Afficher la description des données
print("Description of dataset: ", diabetes_dataset['DESCR'])
```

Description of dataset: `.. _diabetes_dataset:`

Diabetes dataset

-----

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of  $n = 442$  diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- age      age in years
- sex
- bmi      body mass index
- bp      average blood pressure
- s1      tc, total serum cholesterol
- s2      ldl, low-density lipoproteins
- s3      hdl, high-density lipoproteins
- s4      tch, total cholesterol / HDL
- s5      ltg, possibly log of serum triglycerides level
- s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of ``n_samples`` (i.e. the sum of squares of each column totals 1).

Source URL:

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>

For more information see:

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," *Annals of Statistics* (with discussion), 407-499.

([https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle\\_2002.pdf](https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf))

---

## Jeu de données d'apprentissage et Jeu de données de test

Diviser 'data' et 'target' en deux jeux de données pour l'apprentissage et le test (par défaut 75-25%)

```
<code>
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)
</code>
```

```
In [ ]: from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(diabetes_dataset['data'],diabetes_dataset['target'],
```

---

## Importer les classes des modèles de regression

```
<code>
    from sklearn.module_name import model_class
</code>
```

```
In [ ]: from sklearn.neighbors import KNeighborsRegressor
        from sklearn.linear_model import LinearRegression, Ridge, Lasso
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.svm import SVR, LinearSVR
        from sklearn.neural_network import MLPRegressor
```

---

## Instancier les modèles

```
<code>
    model = model_class()
</code>
```

```
In [ ]: knnr1 = KNeighborsRegressor(n_neighbors = 1)
        knnr5 = KNeighborsRegressor(n_neighbors = 5)
        lir = LinearRegression()
        ridge1 = Ridge(alpha=1)
        ridge10 = Ridge(alpha=10)
        lasso1 = Lasso(alpha=1)
        lasso10 = Lasso(alpha=10)
        dtr3 = DecisionTreeRegressor(max_depth = 3)
        dtr10 = DecisionTreeRegressor(max_depth = 10)
        rfr3 = RandomForestRegressor(n_estimators = 3)
        rfr10 = RandomForestRegressor(n_estimators = 10)
        svr10_10 = SVR(C=10,gamma=10)
        svr1_10 = SVR(C=1,gamma=10)
        mplr10 = MLPRegressor(solver='lbfgs',hidden_layer_sizes=[10], max_iter= 10000)
        mplr10_10 = MLPRegressor(solver='lbfgs',hidden_layer_sizes=[10], max_iter=10000)
        for model in [knnr1,knnr5,lir,ridge1,ridge10,lasso1,lasso10,dtr3,dtr10,rfr3,rfr10,svr10_10,svr1_10]:
            print(model,"\n")
```



```

KNeighborsRegressor(n_neighbors=1)

KNeighborsRegressor()

LinearRegression()

Ridge(alpha=1)

Ridge(alpha=10)

Lasso(alpha=1)

Lasso(alpha=10)

DecisionTreeRegressor(max_depth=3)

DecisionTreeRegressor(max_depth=10)

RandomForestRegressor(n_estimators=3)

RandomForestRegressor(n_estimators=10)

SVR(C=10, gamma=10)

SVR(C=1, gamma=10)

MLPRegressor(hidden_layer_sizes=[10], max_iter=10000, solver='lbfgs')

MLPRegressor(hidden_layer_sizes=[10], max_iter=10000, solver='lbfgs')

```

---

## Entraînement des modèles sur le jeu de données d'entraînement

```

<code>
    model.fit(X_train,y_train)
</code>

```

## Évaluation des prédictions sur les jeux de données d'entrainement et de test

```

<code>
    model.score(X_train,y_train)
    model.score(X_test,y_test)
</code>

```

```

In [ ]: print("Model\t Train set score", "\tTest set score: ", "\n\n-----")
for model in [knnr1,knnr5,lir,ridge1,ridge10,lasso1,lasso10,dtr3,dtr10,rfr3,rfr10,svr10_10,svr1_10]:
    model.fit(X_train,y_train)
    print(type(model).__name__, "\t", model.score(X_train,y_train), "\t", model.score(X_test,y_test))
    print('-----')

```

Model	Train set score	Test set score:
-------	-----------------	-----------------

KNeighborsRegressor	1.0	-0.5549498958090489
---------------------	-----	---------------------

KNeighborsRegressor	0.6238758317268587	0.18912404854026388
---------------------	--------------------	---------------------

LinearRegression	0.5554337250189862	0.35940880381777107
------------------	--------------------	---------------------

Ridge	0.4625420370188341	0.3569596077458861
-------	--------------------	--------------------

Ridge	0.17131356091679695	0.14333099992172604
-------	---------------------	---------------------

Lasso	0.41412350907212847	0.27818075944340903
-------	---------------------	---------------------

Lasso	0.0	-0.00014359578276068596
-------	-----	-------------------------

DecisionTreeRegressor	0.5775372023714627	0.08676750963774293
-----------------------	--------------------	---------------------

DecisionTreeRegressor	0.9447464021555244	-0.09367002671526925
-----------------------	--------------------	----------------------

RandomForestRegressor	0.8634572764128708	0.058868269046006616
-----------------------	--------------------	----------------------

RandomForestRegressor	0.8964358112080978	0.18158870568778274
-----------------------	--------------------	---------------------

SVR	0.5442466909223621	0.3892437574477601
-----	--------------------	--------------------

SVR	0.16889879397988272	0.12734739650103954
-----	---------------------	---------------------

MLPRegressor	0.6004639794759812	0.34598162610303496
--------------	--------------------	---------------------

MLPRegressor	0.6153888199527111	0.3436647261991872
--------------	--------------------	--------------------

---

## Prédire avec un modèle entraîné

```
<code>
    y_pred = model.predict(X_test)
</code>
```

```
In [ ]: y_pred = lir.predict(X_test)
        print(X_test.shape,y_pred.shape)
```

```
(111, 10) (111,)
```

```
In [ ]: # Afficher les valeurs prédites
        print("y predict pour X_test: ", y_pred)
```

```
y predict pour X_test: [241.84546349 250.11946624 164.96775859 119.11621898 188.23195881
260.55946852 113.07808671 190.54266768 151.88724196 236.50507432
168.76762119 180.52581698 109.16306789 90.19978477 244.73716844
90.57897053 152.50836118 66.97673392 98.0432504 215.3968854
197.70918729 160.91650284 162.88667688 158.25269476 202.44790389
168.46930106 119.87054445 83.05807801 189.98101417 163.02091883
177.07989723 82.66965767 144.53256644 146.07829878 141.7377238
195.18646525 164.18028205 189.14832914 128.13180502 206.12545226
82.64160932 164.94613662 144.46101578 182.0579875 178.41206706
72.55434005 142.6977982 140.44004414 121.74956963 233.70582676
162.07562213 76.90068953 155.68723639 156.63966204 238.10941527
175.75890957 190.82215073 119.48016597 131.31514527 172.24751733
214.44182872 171.30823418 156.69389424 110.97544275 257.79364104
154.64914571 81.1062021 227.26595089 205.44747738 46.92211004
78.27931883 131.53378104 105.63686403 145.15936864 133.85348634
189.31345673 99.29993167 202.16453375 221.67699238 189.36629355
150.25464898 208.85466761 48.0354776 206.61918247 76.63042511
92.70280795 148.19314939 194.59785954 133.08269028 148.56783463
97.5125683 124.56623148 82.49281617 151.42006291 124.81687754
105.29225265 236.21122712 227.49113367 128.37813597 164.29503518
193.35369308 111.84052153 204.36737503 84.74273809 217.69919582
113.61428273 221.22291819 267.69564687 115.62346943 113.98277339
195.01210236]
```

---

## 4 - Choix du jeu de données : Jeu de données 'breast\_cancer' pour la classification binaire

Lien: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

[learn.org/stable/modules/generated/sklearn.datasets.load\\_breast\\_cancer.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

---

```
In [ ]: # Format du jeu de données
        print(breast_cancer_dataset['data'].shape,breast_cancer_dataset['target'].shape)
```

```
(569, 30) (569,)
```

---

## Jeu de données d'apprentissage et Jeu de données de test

```
In [ ]: X_train,X_test,y_train,y_test = train_test_split(breast_cancer_dataset['data'],breast_cancer_dat
```

---

## Importer les classes des modèles de classification

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.linear_model import LogisticRegression
```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neural_network import MLPClassifier

```

## Instancier les modèles

```

In [ ]: knnc1 = KNeighborsClassifier(n_neighbors = 1)
knnc5 = KNeighborsClassifier(n_neighbors = 5)
lor = LogisticRegression(C=100, max_iter=15000)
dtc3 = DecisionTreeClassifier(max_depth = 3)
dtc10 = DecisionTreeClassifier(max_depth = 10)
rfc3 = RandomForestClassifier(n_estimators = 3)
rfc10 = RandomForestClassifier(n_estimators = 10)
svc10_10 = SVC(C=10, gamma=10)
svc1_10 = SVC(C=1, gamma=10)
mplc10 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=[10], max_iter=15000)
mplc10_10 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=[10], max_iter=15000)
for model in [knnc1, knnc5, lor, dtc3, dtc10, rfc3, rfc10, svc10_10, svc1_10, mplc10, mplc10_10]:
    print(model, "\n")

```

KNeighborsClassifier(n\_neighbors=1)

KNeighborsClassifier()

LogisticRegression(C=100, max\_iter=15000)

DecisionTreeClassifier(max\_depth=3)

DecisionTreeClassifier(max\_depth=10)

RandomForestClassifier(n\_estimators=3)

RandomForestClassifier(n\_estimators=10)

SVC(C=10, gamma=10)

SVC(C=1, gamma=10)

MLPClassifier(hidden\_layer\_sizes=[10], max\_iter=15000, solver='lbfgs')

MLPClassifier(hidden\_layer\_sizes=[10], max\_iter=15000, solver='lbfgs')

## Entraînement des modèles sur le jeu de données d'entraînement

### Évaluation des prédictions sur les jeux de données d'entraînement et de test

```

In [ ]: for model in [knnc1, knnc5, lor, dtc3, dtc10, rfc3, rfc10, svc10_10, svc1_10, mplc10, mplc10_10]:
        model.fit(X_train, y_train)
        print(type(model).__name__, ": Train set score: ", model.score(X_train, y_train), "Test set sco

```

KNeighborsClassifier : Train set score: 1.0 Test set score: 0.916083916083916

KNeighborsClassifier : Train set score: 0.9413145539906104 Test set score: 0.9370629370629371

LogisticRegression : Train set score: 0.9788732394366197 Test set score: 0.958041958041958

DecisionTreeClassifier : Train set score: 0.9765258215962441 Test set score: 0.916083916083916

DecisionTreeClassifier : Train set score: 1.0 Test set score: 0.8951048951048951

RandomForestClassifier : Train set score: 0.9741784037558685 Test set score: 0.958041958041958

RandomForestClassifier : Train set score: 1.0 Test set score: 0.965034965034965

SVC : Train set score: 1.0 Test set score: 0.6293706293706294

SVC : Train set score: 1.0 Test set score: 0.6293706293706294

MLPClassifier : Train set score: 0.6267605633802817 Test set score: 0.6293706293706294

MLPClassifier : Train set score: 0.9436619718309859 Test set score: 0.9440559440559441

## Calcul direct du score avec la fonction 'mean' de np

```
In [ ]: y_train_pred = knnc1.predict(X_train)
        y_test_pred = knnc1.predict(X_test)
        print(type(knnc1).__name__, ": Train set score: ", np.mean(y_train_pred==y_train), "Test set score
```

KNeighborsClassifier : Train set score: 1.0 Test set score: 0.916083916083916

---