

## What is Scikit-Learn (Sklearn)

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon **NumPy**, **SciPy** and **Matplotlib**.

## Origin of Scikit-Learn

It was originally called ***scikits.learn*** and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation), took this project at another level and made the first public release (v0.1 beta) on 1st Feb. 2010.

## Features

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows –

**Supervised Learning algorithms** – Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

**Unsupervised Learning algorithms** – On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

**Clustering** – This model is used for grouping unlabelled data.

**Cross Validation** – It is used to check the accuracy of supervised models on unseen data.

**Dimensionality Reduction** – It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.

**Ensemble methods** – As name suggest, it is used for combining the predictions of multiple supervised models.

**Feature extraction** – It is used to extract the features from data to define the attributes in image and text data.

**Feature selection** – It is used to identify useful attributes to create supervised models.

**Open Source** – It is open-source library and also commercially usable under BSD license.

## Example

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of X:\n", X[:10])
```

**Scikit-learn** is a machine learning library for Python. It features several regression, classification and clustering algorithms including SVMs, gradient boosting, k-means, random forests and DBSCAN. It is designed to work with Python Numpy and SciPy. The scikit-learn project kicked off as a Google Summer of Code (also known as GSoC) project by David Cournapeau as scikits.learn. It gets its name from “Scikit”, a separate third-party extension to SciPy.

## Python Scikit-learn

Scikit is written in Python (most of it) and some of its core algorithms are written in Cython for even better performance. Scikit-learn is used to build models and it is not recommended to use it for reading, manipulating and summarizing data as there are better frameworks available for the purpose. It is open source and released under BSD license.

### [Install Scikit Learn](#)

Scikit assumes you have a running Python 2.7 or above platform with NumPY (1.8.2 and above) and SciPY (0.13.3 and above) packages on your device. Once we have these packages installed we can proceed with the installation. For [pip](#) installation, run the following command in the terminal:

```
pip install scikit-learn
```

If you like conda, you can also use the conda for package installation, run the following command:

```
conda install scikit-learn
```

### [Using Scikit-Learn](#)

Once you are done with the installation, you can use scikit-learn easily in your Python code by importing it as:

```
import sklearn
```

### [Scikit Learn Loading Dataset](#)

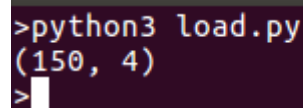
Let's start with loading a dataset to play with. Let's load a simple dataset named Iris. It is a dataset of a flower, it contains 150 observations about different measurements of the flower. Let's see how to load the dataset using scikit-learn.

```
# Import scikit learn
from sklearn import datasets

# Load data
iris= datasets.load_iris()

# Print shape of data to confirm data is loaded
print(iris.data.shape)
```

We are printing shape of data for ease, you can also print whole data if you wish so, running the codes gives



```
>python3 load.py
(150, 4)
>
```

an output like this:

### [Scikit Learn SVM - Learning and Predicting](#)

Now we have loaded data, let's try learning from it and predict on new data. For this purpose we have to create an estimator and then call its fit method.

```
from sklearn import svm
from sklearn import datasets

# Load dataset
iris = datasets.load_iris()

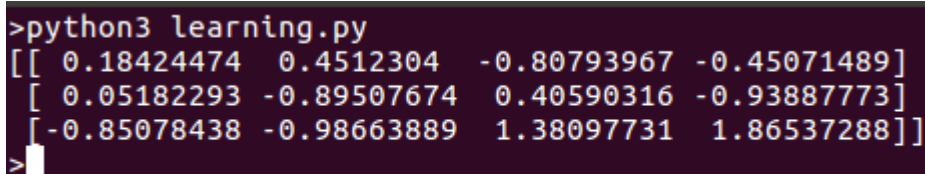
clf = svm.LinearSVC()

# learn from the data
clf.fit(iris.data, iris.target)

# predict for unseen data
clf.predict([[ 5.0, 3.6, 1.3, 0.25]])

# Parameters of model can be changed by using the attributes ending with an underscore
print(clf.coef_)
```

Here is what we get when we run this script:



```
>python3 learning.py
[[ 0.18424474  0.4512304 -0.80793967 -0.45071489]
 [ 0.05182293 -0.89507674  0.40590316 -0.93887773]
 [-0.85078438 -0.98663889  1.38097731  1.86537288]]
>
```

## [Scikit Learn Linear Regression](#)

Creating various models is rather simple using scikit-learn. Let's start with a simple example of regression.

```
#import the model

from sklearn import linear_model

reg = linear_model.LinearRegression()

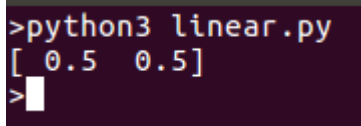
# use it to fit a data

reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

# Let's look into the fitted data

print(reg.coef_)
```

Running the model should return a point that can be plotted on the same line:



```
>python3 linear.py
[ 0.5  0.5]
>
```

## [k-Nearest neighbour classifier](#)

Let's try a simple classification algorithm. This classifier uses an algorithm based on ball trees to represent the training samples.

```
from sklearn import datasets

# Load dataset

iris = datasets.load_iris()

# Create and fit a nearest-neighbor classifier

from sklearn import neighbors

knn = neighbors.KNeighborsClassifier()

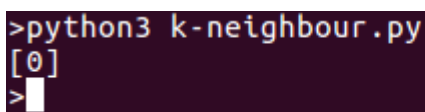
knn.fit(iris.data, iris.target)

# Predict and print the result

result=knn.predict([[0.1, 0.2, 0.3, 0.4]])

print(result)
```

Let's run the classifier and check results, the classifier should return 0. Let's try the example:



```
>python3 k-neighbour.py
[0]
>
```

## [K-means clustering](#)

This is the simplest clustering algorithm. The set is divided into 'k' clusters and each observation is assigned to a cluster. This is done iteratively until the clusters converge. We will create one such clustering model in the following program:

```
from sklearn import cluster, datasets

# load data

iris = datasets.load_iris()

# create clusters for k=3

k=3

k_means = cluster.KMeans(k)

# fit data

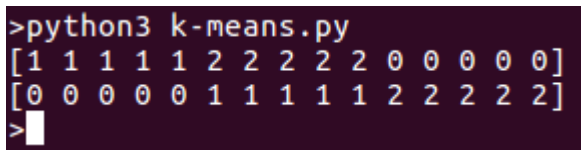
k_means.fit(iris.data)

# print results

print( k_means.labels_[:10])

print( iris.target[:10])
```

On running the program we'll see separate clusters in the list. Here is the output for above code snippet:



```
>python3 k-means.py
[1 1 1 1 1 2 2 2 2 2 0 0 0 0 0]
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
>
```

---

## SciPy

**What is SciPy?**

**SciPy is a scientific computation library that uses NumPy underneath.**

**SciPy stands for Scientific Python.**

**It provides more utility functions for optimization, stats and signal processing.**

**Like NumPy, SciPy is open source so we can use it freely.**

**SciPy was created by NumPy's creator Travis Olliphant.**

---

**Why Use SciPy?**

If SciPy uses NumPy underneath, why can we not just use NumPy?

SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

---

Which Language is SciPy Written in?

SciPy is predominantly written in Python, but a few segments are written in C.

Constants in SciPy

As SciPy is more focused on scientific implementations, it provides many built-in scientific constants.

These constants can be helpful when you are working with Data Science.

PI is an example of a scientific constant.

Print the constant value of PI:

```
from scipy import constants
```

```
print(constants.pi)
```

---

Constant Units

A list of all units under the constants module can be seen using the dir() function.

Example

List all constants:

```
from scipy import constants
```

```
print(dir(constants))
```

---

Unit Categories

The units are placed under these categories:

- Metric
- Binary
- Mass
- Angle
- Time
- Length

- Pressure
- Volume
- Speed
- Temperature
- Energy
- Power
- Force

```
from scipy.optimize import root
from math import cos
```

```
def eqn(x):
    return x + cos(x)
```

```
myroot = root(eqn, 0)
```

```
print(myroot.x)
```

```
-----
```

```
from scipy.optimize import minimize
```

```
def eqn(x):
    return x**2 + x + 2
```

```
mymin = minimize(eqn, 0, method='BFGS')
```

```
print(mymin)
```

```
-----
```