SCIKIT-LEARN

Machine Learning Library

Presented By :-

Aadarsh Yadav

Aditya Pratap Singh

Devesh Soni

WHAT IS SCIKIT-LEARN (SKLEARN)

- Scikit-learn (Sklearn) is one of the most useful 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 consistence 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.

INSTALLATION

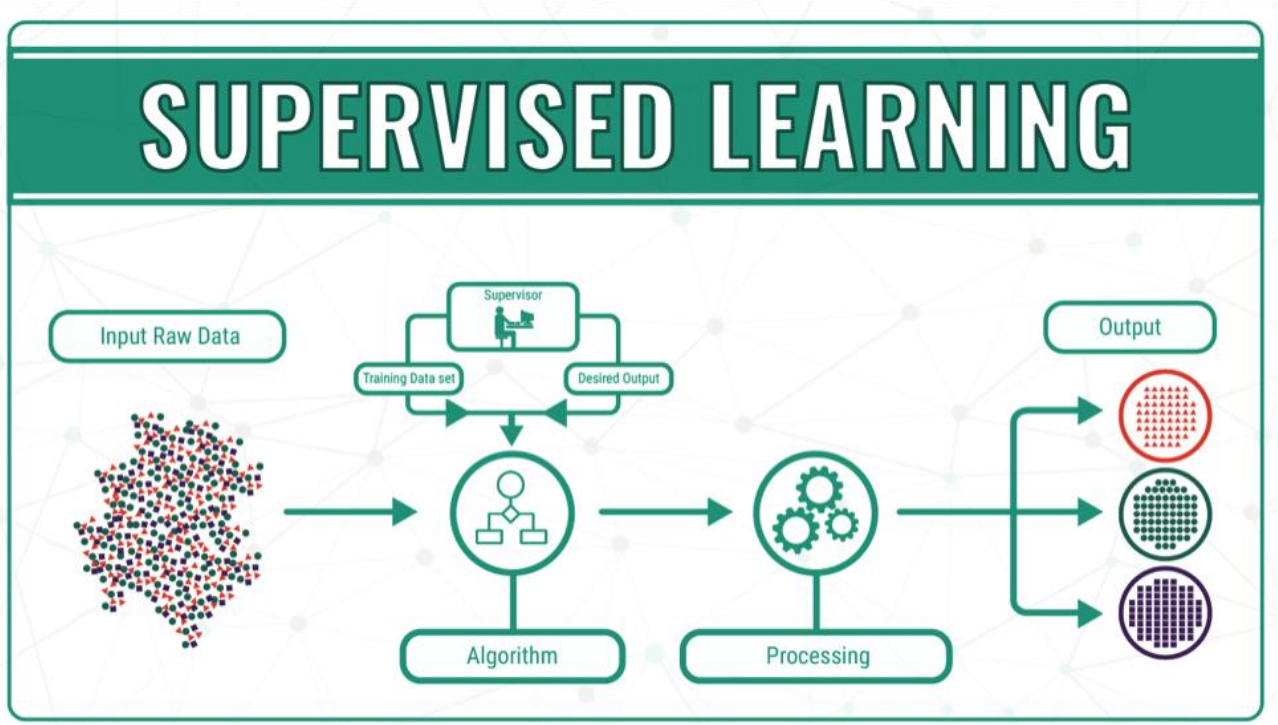
- **Using pip**
- Following command can be used to install scikit-learn via pip
- `pip install -U scikit-learn`

- **Using conda**
- Following command can be used to install scikit-learn via conda
- `conda install scikit-learn`

FEATURES

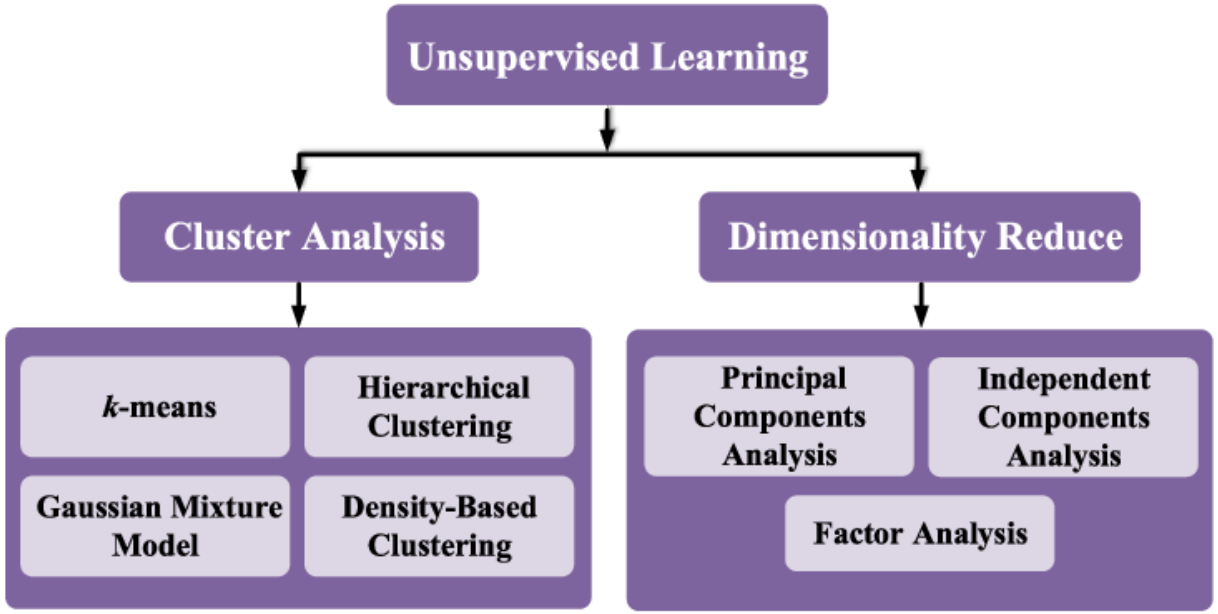
- Rather than focusing on loading, manipulating and summarizing data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are up next.

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.

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 summarization, visualization 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.

SCIKIT LEARN - MODELLING PROCESS

1. DATASET LOADING

- A collection of data is called dataset. It is having the following two components −
- **Features** − The variables of data are called its features. They are also known as predictors, inputs or attributes.
- **Response** − It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

Example

Following is an example to load **iris** dataset −

```
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])
```

Output

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
First 10 rows of X:
[
   [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]
]
```

2. SPLITTING THE DATASET

- To check the accuracy of our model, we can split the dataset into two pieces - **a training set** and **a testing set**. Use the training set to train the model and testing set to test the model. After that, we can evaluate how well our model did.

Example

The following example will split the data into 70:30 ratio, i.e. 70% data will be used as training data and 30% will be used as testing data. The dataset is iris dataset as in above example.

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size = 0.3, random_state = 1
)

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

Output

```
(105, 4)
(45, 4)
(105,)
(45,)
```

3. TRAIN THE MODEL

- Next, we can use our dataset to train some prediction-model. As discussed, scikit-learn has wide range of **Machine Learning (ML) algorithms** which have a consistent interface for fitting, predicting accuracy, recall etc.

Example

In the example below, we are going to use KNN (K nearest neighbors) classifier. Don't go into the details of KNN algorithms, as there will be a separate chapter for that. This example is used to make you understand the implementation part only.

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size = 0.4, random_state=1
)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
classifier_knn = KNeighborsClassifier(n_neighbors = 3)
classifier_knn.fit(X_train, y_train)
y_pred = classifier_knn.predict(X_test)
# Finding accuracy by comparing actual response values(y_test)with predicted response value(y_pred)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
# Providing sample data and the model will make prediction out of that data

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]
preds = classifier_knn.predict(sample)
pred_species = [iris.target_names[p] for p in preds] print("Predictions:", pred_species)
```

Output

```
Accuracy: 0.9833333333333333
Predictions: ['versicolor', 'virginica']
```

4. PREPROCESSING THE DATA

- As we are dealing with lots of data and that data is in raw form, before inputting that data to machine learning algorithms, we need to convert it into meaningful data. This process is called preprocessing the data. Scikit-learn has package named **preprocessing** for this purpose.
- The **preprocessing** package has the following techniques −
- Binarization
- Mean Removal
- Scaling
- Normalization

Binarisation

This preprocessing technique is used when we need to convert our numerical values into Boolean values.

Example

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [2.1, -1.9, 5.5],
   [-1.5, 2.4, 3.5],
   [0.5, -7.9, 5.6],
   [5.9, 2.3, -5.8]]
)
data_binarized = preprocessing.Binarizer(threshold=0.5).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

In the above example, we used **threshold value** = 0.5 and that is why, all the values above 0.5 would be converted to 1, and all the values below 0.5 would be converted to 0.

Output

```
Binarized data:
[
   [ 1. 0. 1.]
   [ 0. 1. 1.]
   [ 0. 0. 1.]
   [ 1. 1. 0.]
]
```

Mean Removal

This technique is used to eliminate the mean from feature vector so that every feature centered on zero.

Example

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [2.1, -1.9, 5.5],
   [-1.5, 2.4, 3.5],
   [0.5, -7.9, 5.6],
   [5.9, 2.3, -5.8]]
)

#displaying the mean and the standard deviation of the input data
print("Mean =", input_data.mean(axis=0))
print("Stddeviation = ", input_data.std(axis=0))
#Removing the mean and the standard deviation of the input data

data_scaled = preprocessing.scale(input_data)
print("Mean_removed =", data_scaled.mean(axis=0))
print("Stddeviation_removed =", data_scaled.std(axis=0))
```

Output

```
Mean = [ 1.75 -1.275 2.2 ]
Stddeviation = [ 2.71431391 4.20022321 4.69414529]
Mean_removed = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]
Stddeviation_removed = [ 1. 1. 1.]
```

Scaling

We use this preprocessing technique for scaling the feature vectors. Scaling of feature vectors is important, because the features should not be synthetically large or small.

Example

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [
      [2.1, -1.9, 5.5],
      [-1.5, 2.4, 3.5],
      [0.5, -7.9, 5.6],
      [5.9, 2.3, -5.8]
   ]
)
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print ("\nMin max scaled data:\n", data_scaled_minmax)
```

Output

```
Min max scaled data:
[
   [ 0.48648649 0.58252427 0.99122807]
   [ 0. 1. 0.81578947]
   [ 0.27027027 0. 1. ]
   [ 1. 0.99029126 0. ]
]
```

Normalisation

We use this preprocessing technique for modifying the feature vectors. Normalisation of feature vectors is necessary so that the feature vectors can be measured at common scale. There are two types of normalisation as follows –

L1 Normalisation

It is also called Least Absolute Deviations. It modifies the value in such a manner that the sum of the absolute values remains always up to 1 in each row. Following example shows the implementation of L1 normalisation on input data.

Example

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [
      [2.1, -1.9, 5.5],
      [-1.5, 2.4, 3.5],
      [0.5, -7.9, 5.6],
      [5.9, 2.3, -5.8]
   ]
)
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
print("\nL1 normalized data:\n", data_normalized_l1)
```

Output

```
L1 normalized data:
[
   [ 0.22105263 -0.2 0.57894737]
   [-0.2027027 0.32432432 0.47297297]
   [ 0.03571429 -0.56428571 0.4 ]
   [ 0.42142857 0.16428571 -0.41428571]
]
```

L2 Normalisation

Also called Least Squares. It modifies the value in such a manner that the sum of the squares remains always up to 1 in each row. Following example shows the implementation of L2 normalisation on input data.

Example

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [
      [2.1, -1.9, 5.5],
      [-1.5, 2.4, 3.5],
      [0.5, -7.9, 5.6],
      [5.9, 2.3, -5.8]
   ]
)
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nL1 normalized data:\n", data_normalized_l2)
```

Output

```
L2 normalized data:
[
   [ 0.33946114 -0.30713151 0.88906489]
   [-0.33325106 0.53320169 0.7775858 ]
   [ 0.05156558 -0.81473612 0.57753446]
   [ 0.68706914 0.26784051 -0.6754239 ]
]
```

SCIKIT LEARN - DATA REPRESENTATION

1. DATA AS TABLE

- The best way to represent data in Scikit-learn is in the form of tables. A table represents a 2-D grid of data where rows represent the individual elements of the dataset and the columns represents the quantities related to those individual elements.

Example

With the example given below, we can download ***iris dataset*** in the form of a Pandas DataFrame with the help of python ***seaborn*** library.

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

Output

```
sepal_length sepal_width petal_length petal_width  species
0         5.1         3.5          1.4           0.2   setosa
1         4.9         3.0          1.4           0.2   setosa
2         4.7         3.2          1.3           0.2   setosa
3         4.6         3.1          1.5           0.2   setosa
4         5.0         3.6          1.4           0.2   setosa
```

2. DATA AS FEATURE MATRIX

- Features matrix may be defined as the table layout where information can be thought of as a 2-D matrix. It is stored in a variable named **X** and assumed to be two dimensional with shape [n_samples, n_features]. Mostly, it is contained in a NumPy array or a Pandas DataFrame. As told earlier, the samples always represent the individual objects described by the dataset and the features represents the distinct observations that describe each sample in a quantitative manner.

3. DATA AS TARGET ARRAY

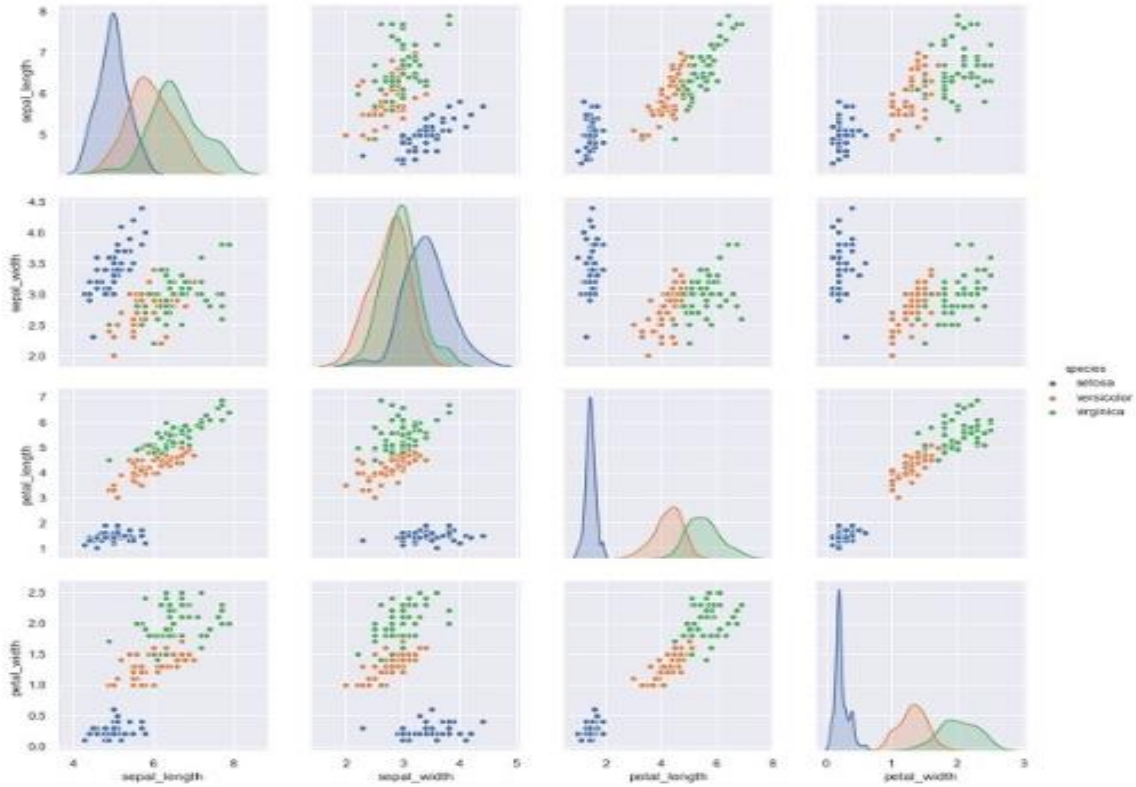
- Along with Features matrix, denoted by X, we also have target array. It is also called label. It is denoted by y. The label or target array is usually one-dimensional having length n_samples. It is generally contained in NumPy ***array*** or Pandas ***Series***. Target array may have both the values, continuous numerical values and discrete values.

Example

In the example below, from iris dataset we predict the species of flower based on the other measurements. In this case, the Species column would be considered as the feature.

```
import seaborn as sns
iris = sns.load_dataset('iris')
%matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris, hue='species', height=3);
```

Output




```
X_iris = iris.drop('species', axis=1)
X_iris.shape
y_iris = iris['species']
y_iris.shape
```

Output

```
(150,4)
(150,)
```