**Traditional machine learning algorithms**

**Support vector machine (SVM):** SVMs are a set of supervised learning methods used for classification, regression, and outliers detection. The python package “sklearn” provides a set of methods to access SVMs. SVC, NuSVC, and LinearSVC are classes capable of performing binary and multi-class classification on a given dataset.

Example 1:

 # This is a single line comment.

# Text followed by ‘#’ in a line is comment

# A multiline comment starts with ''' and ends with '''

'''

This is a multi line comment.

This line is also a comment

'''

#import svm (support vector machine) from sklearn package

from sklearn import svm

#a simple dataset

X = [[0, 1],[1, 2],[2, 1],[2, 3],[1, 3],[2, 2]]

#labels of the dataset X

y = ['a', 'a', 'b', 'b', 'a', 'b']

#create a SVM classifier (i.e., a model) using svm.SVC()

clf = svm.SVC()

#train the model

clf.fit(X, y)

#predict the label of (3, 1)

result1 = clf.predict([[3, 1]])

#print the predicted label of (3, 1)

print(result1)

#predict the label of (0, 2)

result2 = clf.predict([[0, 2]])

#print the predicted label of (0, 2)

print(result2)

#Text in red font is the output of the program code

['b']

['a']

Example 2:

from sklearn import svm

#import dataset

from sklearn.datasets import load\_iris

#iris dataset contains 150 samples, each has 4 features

X, y = load\_iris(return\_X\_y = True)

'''

Parameter ‘return\_X\_y = True’ is required in load\_iris()function to get the sample and label data in seperate variables.

'''

#print the size of sample

print("The size of the sample:", X.shape)

#print first 5 samples

print("First 5 samples:\n", X[0:5])

print("First 5 labels:\n", y[0:5])

#create model

clf = svm.SVC()

#train the model

clf.fit(X, y)

#predict the labels for 45th to 55th samples

result = clf.predict(X[45:55])

#print the predicted lables

print("Predicted labels\n",result)

#print the actual labels of 45th to 55th samples

print("Actual labels\n",y[45:55])

The size of the sample: (150, 4)

First 5 samples:

 [[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]]

First 5 labels:

 [0 0 0 0 0]

Predicted labels

 [0 0 0 0 0 1 1 1 1 1]

Actual labels

 [0 0 0 0 0 1 1 1 1 1]

Example 3: SVM supports various kernels as follows.

#linear kernel

linear\_svc = svm.SVC(kernel=’linear’)

'''

kernel = {’linear’, ’poly’, ’rbf’, ’sigmoid’, ’precomputed’}, default=’rbf’

'''

#Radial Basis Function (RBF) kernel

rbf\_svc = svm.SVC(kernel=’rbf’)

#polynoimial kernel

poly\_svc = svm.SVC(kernel=’poly’, degree =3)

# degree =3 represent the third-degree polynomial

# user defined kernel

import numpy as np

#define a kernal

def my\_kernel(X, Y):

 return np.dot (X, Y.T)

#using the user defined kernel

clf = svm.SVC(kernel= my\_kernel)

**Decision Tree:** It is a supervised learning method. A decision tree creates a simple model which predicts the value of a target variable by learning simple decision rules derived from the data features

Example 1: Simple data set

from sklearn import tree

#a simple dataset

X = [[0, 1],[1, 2],[2, 1],[2, 3],[1, 3],[2, 2]]

# labels of the dataset X

y = ['a', 'a', 'b', 'b', 'a', 'b']

#create a model

clf = tree.DecisionTreeClassifier()

#train the model

clf = clf.fit(X, y)

#predice the label

result = clf.predict([[3, 1]])

#print the predicted label

print(result)

#print the constructed decision tree

tree.plot\_tree(clf)

 ['b']



Example 2: Decision tree under a slightly complex dataset.

from sklearn.datasets import load\_iris

from sklearn import tree

#create model

clf = tree.DecisionTreeClassifier()

#load dataset

X, y = load\_iris(return\_X\_y=True)

#train the model

clf = clf.fit(X, y)

#predict the label of first sample

print(clf.predict([X[0]]))

#print decision tree

tree.plot\_tree(clf)

[0]



Example 3: You can also load the iris dataset like

from sklearn.datasets import load\_iris

from sklearn import tree

#loading dataset without ‘return\_X\_y=True’ parameter

myData = load\_iris()

#myData contains both the sample and labels

#print first 5-samples and 5-labels

print("First 5-samples\n",myData.data[:5])

print("First 5-labels\n", myData.target[:5])

#ctreate a model

clf = tree.DecisionTreeClassifier()

#train the model

clf = clf.fit(myData.data, myData.target)

#predict the label of first sample

result = clf.predict([myData.data[0]])

print("predicted label\n", result)

First 5-samples

 [[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]]

First 5-labels

 [0 0 0 0 0]

predicted label

 [0]

**Random Forest:** Random Forest is an estimatorthat fits several decision tree classifiers on various sub-samples of a given dataset and improves the prediction accuracy by averaging the output of all the employed decision tree classifiers.

Example:

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

#load dataset

X, y = load\_iris(return\_X\_y=True)

'''Create a RndomForestClassifier with 20 trees, each having max depth of 2. The depth of the tree reflects the complexity.

'''

clf = RandomForestClassifier(n\_estimators=20, max\_depth=2)

#train the model

clf.fit(X, y)

#predict the label for kth sample

k = 10

result = clf.predict([X[k]])

#print the label

print(result)

[0]

 **K-Means clustering:** It is an unsupervised learning method, which clusters data by separating samples in n groups of equal variances.



Example 1:

from sklearn.cluster import KMeans

import numpy as np

#a sample dataset

X = np.array([[1, 2], [1, 4], [1, 0],

 [10, 2], [10, 4], [10, 0]])

#create a model for 2 clusters

kmeans = KMeans(n\_clusters=2)

#train the model

kmeans.fit(X)

#print labels

print("Labels:\n",kmeans.labels\_)

#predict the cluster

results = kmeans.predict([[0, 5], [12, 3]])

#print the predicted clusters

print("Predicted clusters:\n", results)

#print the cluster centers

print("Cluster centers:\n", kmeans.cluster\_centers\_)

Labels:

 [1 1 1 0 0 0]

Predicted clusters:

 [1 0]

Cluster centers:

 [[10. 2.]

 [1. 2.]]

**k-nearest neighbors(k-NN):** It is a neighbors-based classification, which attempts to construct a general model by simply storing the instances of the training data. The cluster of a test data is determined by a simple majority vote of the k nearest neighbors of each point.

Example 1: Number of neighbors to use (n\_neighbors = 3)

#sample dataset

X = [[0], [1], [2], [3]]

#label of dataset X(only two labels)

y = [0, 0, 1, 1]

from sklearn.neighbors import KNeighborsClassifier

#create K-NN model with 3 nearest neighbors

neigh = KNeighborsClassifier(n\_neighbors=3)

#train the model

neigh.fit(X, y)

#predict the label

result = neigh.predict([[2.9]])

#print the prediction

print(neigh.predict([[2.9]]))

'''

The model object (i.e., neigh) also contains the “predict\_probability()” function which reflect the probability of the predicted label corresponding to particular category.

'''

#get the label prediction probability

prob = neigh.predict\_proba([[2.9]])

#print label prediction probability

print(prob)

[1]

[[0.33333333 0.66666667]]

With n\_neighbors = 4, the output of the above program code become

[0]

[[0.5 0.5]]

Example 2:

#sample dataset

X = [[0,0], [1,0], [2,1], [3,2], [0,4], [5,1], [1,6]]

#label of dataset X

y = [0, 1, 2, 4, 4, 3, 2]

from sklearn.neighbors import KNeighborsClassifier

#create K-NN model with 5 nearest neighbors

neigh = KNeighborsClassifier(n\_neighbors=5)

#train the model

neigh.fit(X, y)

#predict the label

result = neigh.predict([[1,2]])

#print the prediction

print(result)

#label prediction probability

prob = neigh.predict\_proba([[1,2]]))

#print label prediction probability

print(prob)

[4]

[[0.2 0.2 0.2 0. 0.4]]