16-Scikit Learn- Boosting Methods

Boosting methods build ensemble model in an increment way. The main principle is to build the model incrementally by training each base model estimator sequentially. In other to build powerful ensemble, these methods basically combine several week learners which are sequentially trained over multiple iterations of training data. Sklearn ensemble module having following two boosting methods:

16-1-Adaboost: It is one of the most successful boosting ensemble method whose main key is in the way they give weights to the instances in dataset.

- **Classification with Adaboost:** Sklearn module provides **sklearn.ensemble.AdaBoostClassifier.** The main parameter this module use is **base estimator.** base_estimator is is the value of the base estimator from which the boosted ensemble is built. if choose this parameter's value to none, the base estimator would be DecisionTreeClassifier.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
X,y=make_classification(n_samples=1000,n_features=10)
ADBclf=AdaBoostClassifier(n_estimators=100,random_state=0)
ADBclf.fit(X,y)
print(ADBclf.predict([[0,2,6,9,8,4,2,3,5,3]]))
```

output:

[1]

- **Regression with AdaBoost:** Scikit learn provides **sklearn.ensemble.AdaBoostRegressor.** it will use the same parameters as used by sklearn.ensemble.AdaBoostClassifier.

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import make_classification
X,y=make_classification(n_samples=1000,n_features=10)
ADBclf=AdaBoostRegressor(n_estimators=100,random_state=0)
ADBclf.fit(X,y)
print(ADBclf.predict([[0,2,6,9,8,4,2,3,5,3]]))
```

output:

[0.63363363]

16-2- Gradient Tree Boosting: It is basically a generalization of boosting to arbitrary differentiable loss functions. It produces a prediction model in the form of an ensemble of week prediction models. It can be used for the regression and classification problems. Their main advantage lies in the fact that they naturally handle the mixed type data.

- **Classification with Gradient Tree Boost:** Scikit Learn module provides **sklearn.ensemble.GradientBoostingClassifier**. the main parameter this module use is **loss**.'loss' is the value of loss function to be optimized.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_hastie_10_2
X,y=make_hastie_10_2(random_state=0)
X_train,X_test=X[:5000],X[5000:]
y_train,y_test=y[:5000],y[5000:]
GDBclf=GradientBoostingClassifier(n_estimators=50,learning_rate = 1.0,max_depth = 1, random_state = 0).fit(X_train,y_train)
print(GDBclf.score(X_test,y_test))
```

output:

0.872

- **Regression with Gradient Boosting:** Scikit-learn provides **sklearn.ensemble.GradientBoostingRegressor**.

```
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_friedman1
from sklearn.ensemble import GradientBoostingRegressor
X,y=make_friedman1(n_samples=2000,random_state=0)
X_train,X_test=X[:1000],X[1000:]
y_train,y_test=y[:1000],y[1000:]
GDBreg=GradientBoostingRegressor(n_estimators=80,loss='ls').fit(X_train,y_train)
y_pred=GDBreg.predict(X_test)
print(mean_squared_error(y_test,y_pred))
```

output:

1.491

17- Scikit learn- Clustering Methods

Clustering methods one of the most usual unsupervised ML methods, used to find similarity & relationship patterns among data samples. They cluster those samples into groups having similarity based on features. Scikit learn have **sklearn.cluster** to perform clustering of unlabeled data.

17-1-KMeans

This algorithm computes centroids and iterates until it finds optimal centroid. It requires the number of clusters to be specified that's why it assumes that they are already known. Main of this algorithm is to cluster the data separating samples in n number of groups of equal variances by minimizing the criteria known as the inertia. The number of clusters identified by algorithms is represented by K. Scikit learn have **sklearn.cluster.KMeans**module to perform k-means clustering.

17-2-Affinity Propagation

This algorithm is based on the concept of 'message passing' between different pairs of samples until convergence. It does not require the number of clusters to be specified before running the algorithm. The algorithm has a time complexity of the order O(N2T) which is the biggest disadvantage of it. scikit learn have **sklearn.cluster.AffinityPropagation** module to perform this clustering method.

17-3- Mean Shift

The algorithm mainly discovers blobs in a smooth density of samples. It assigns the datapoints to the clusters iteratively by shifting points towards the highest density of datapoints. Scikit learn have **sklearn.cluster.Meanshift** module to perform Mean Shift Clustering.

17-4- Spectral Clustering

Before clustering, this algorithm basically uses the eigenvalues i.e. spectrum of the similarity matrix of the data to perform dimensionality reduction in fewer dimension. Use of this algorithm is not advisable when there are large number of clusters. scikit learn have **sklearn.cluster.SpectralClustering** module to perform this clustering.

17-5-Hierarchical Clustering

This algorithm builds nested clusters by merging or spliting the clusters successively. It falls into following two categories:

- **Agglomerative hierarchical algorithms:** Here, every data point is treated like a single cluster. It then successively agglomerates the pairs of clusters.
- **Divisive hierarchical algorithms:** In this algorithm, all data points are treated as one big cluster. In this the process of clustering involves dividing, by using top-down approach, the one big cluster into various small clusters. scikit learn have **sklearn.cluster.AgglomerativeClustering** module to perform Agglomerative Hierarchical clustering.

17-6-DBSCAN

It stands for **"Density-based spatial clustering of applications with 'noise'.** This algorithm is based on the intuitive notion of "clusters" & "noise" that clusters are dense regions of the lower density in the data space, separated by lower density regions of data points.Scikit learn have **sklearn.cluster.DBSCAN** module to perform DBSCAN clustering.

17-7-OPTICS

It stands for "ordering points to identity the clustering structure". This algorithm also finds density-based clusters in spatial data. It's basic working logic is like DBSCAN. It addresses a major weakness of DBSCAN algorithm-the problem of detecting meaningful clusters in data of varying density-by ordering the points of the database in such a way that spatially closest points become neighbors in the ordering. scikit learn have **sklearn.cluster.OPTICS** module to perform OPTICS clustering.

17-8-BIRCH

It stands for Balanced iterative reducing and clustering using hierarchies. It builds a tree named **CFT,** for the given data. The advantage of CFT is that the data nodes called CF nodes holds the necessary information for clustering which further prevents the need to hold the entire input data in memory. Scikit learn have **sklearn.cluster.Birch** module to perform BIRCH clustering.

18-Comparing Clustering Algorithms

algorithm	Parameters	Scalability	Metric Used
K-means	Numebr of clusters	Very large n_samples	The distance between points.
Affinity Propagation	Damping	It's not scalable with n_samples	Graph Distance.
Mean-Shift	Bandwidth	It's not scalable with n_samples	The distance between points.
Spectral Clustering	Number of clusters	Medium level of scalability with n_samples. Small level of scalability with n_clusters.	Graph Distance.
Hierarchical Clustering	Distance threshold / Number of clusters	Large n_samples , Large n_clusters	The distance between points.
DBSCAN	Size of neighborhood	Very large n_samples and medium n_clusters.	Nearest point distance.
OPTICS	Minimum cluster membership	Very large n_samples and large n_clusters.	The distance between points.
BIRCH	Threshold/Branching factor	Large n_samples , Large n_clusters	The euclidean distance between points.

18-1- Implementation Example (K-Means Clustering)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns;sns.set()
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from scipy.stats import mode
from sklearn.metrics import accuracy_score
data=load_digits()
kmeans=KMeans(n_clusters=10,random_state=0)
clusters=kmeans.fit_predict(data.data)
for i in range(10):
    mask=(clusters==i)
labels[mask]=mode(data.target[mask])[0]
accuracy_score(data.target,labels)
```

output:

0.155