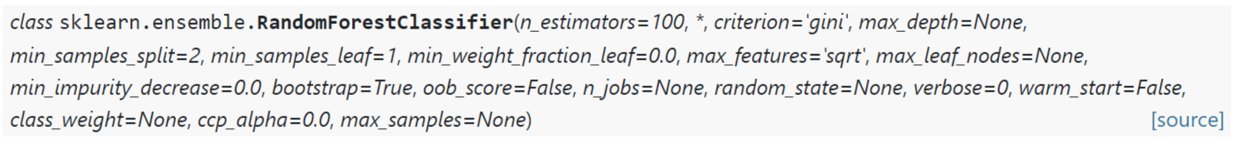
Random Forest

Overview of Random Forest algorithm

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Code Implementation

1. Parameters



```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)                                                              [source]
```

n_ Estimators: the number of trees in the random forest, that is, the number of learners.

max_ Depth: the maximum depth of the tree. If default = none is selected, the tree will expand uniformly until all leaf nodes are of the same type of samples, or the number of minimum sample partitions (min_samples_split) is reached.

min_ samples_ Split: the minimum number of samples divided, that is, if the number of samples is less than or equal to this value, the current node cannot be divided.

n_ Jobs: the number of processes used in parallel. The default is 1. If it is set to - 1, the value is the total number of cores.

random_ State: random state, generated by np.numpy by default.

2. Examples

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
```

```
>>> print(clf.predict([[0, 0, 0, 0]]))
```

Run the program and screenshot the output result:____________________

3. Methods

`apply`(X)	Apply trees in the forest to X, return leaf indices.
`decision_path`(X)	Return the decision path in the forest.
`fit`(X, y[, sample_weight])	Build a forest of trees from the training set (X, y).
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict class for X.
`predict_log_proba`(X)	Predict class log-probabilities for X.
`predict_proba`(X)	Predict class probabilities for X.
`score`(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.

4. Practice

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestClassifier

##Create 100 classes with 10000 samples and 10 features per sample
X, y = make_blobs(n_samples=10000, n_features=10, centers=100,
random_state=0)

## Random forest
clf2 = RandomForestClassifier(n_estimators=10, max_depth=None,
min_samples_split=2, random_state=0)
scores2 = cross_val_score(clf2, X, y)
print(scores2.mean())
```

Run the program and screenshot the output result:____________________

cross_ val_ score(model_name, X,y， cv=k)

Function: verify the stability of a model on a training set and output K prediction accuracy.

K-fold cross validation (k-fold)

The initial training samples are divided into k pieces, of which (k-1) is used as the training set and the remaining one is used as the evaluation set. In this way, the classifier can be trained K times and K training results can be obtained.