Exercise 01:

criterion

In order to transform a table into a tree, the decision tree needs to find the best node and the best branching method. For classification trees, the index to measure the "best" is called "impure". Generally speaking, the lower the impure, the better the decision tree fits the training set. The core of decision tree algorithm in branch method is mostly around the optimization of some impure related index.

In other words, the purity of each parent node in the tree is not necessarily lower than that of a child node.

The parameter criterion is used to determine the calculation method of impurity. Sklearn offers two options:

1) Enter "entropy" to use entropy

2) Enter "Gini" to use Gini coefficient

Where T represents a given node, I represents any classification of tags, and P (i| T) represents the proportion of label classification I to node t. Note that when using information entropy, sklearn actually calculates the information gain based on information entropy, that is, the difference between the information entropy of the parent node and that of the child node.

Compared with Gini coefficient, information entropy is more sensitive to impure, and the punishment of impure is the strongest. But in practice, the effect of information entropy and Gini coefficient is basically the same. The calculation of information entropy is slower because the calculation of Gini coefficient does not involve logarithm. In addition, because the information entropy is more sensitive to the impurity, the growth of the decision tree will be more "refined" when the information entropy is used as an indicator. Therefore, for high-dimensional data or data with a lot of noise, the information entropy is easy to over fit, and the Gini coefficient is often better in this case. Of course, this is not absolute.

The lower the impure, the better the decision tree can fit the training set

random_state

random_ state is used to set the parameters of random patterns in branches. The default value is none. The randomness will be more obvious in high-dimensional data sets. For low-dimensional data sets (such as iris data set), the randomness will hardly show. If you input any integer, the same tree will grow all the time to stabilize the model.

splitter

Splitter is also used to control the random options in the decision tree. There are two kinds of input values, i.e. "best". Although the decision tree branches randomly, it will give priority to the more important features for branching (the importance can be determined by the attribute feature)_ importances_ If you input "random", the decision tree will be more random when branching, the tree will be deeper and larger because it contains more unnecessary information, and the fitting of the training set will be reduced due to the unnecessary information. This is also a way to prevent over fitting. When you predict that your model will over fit, use these two parameters to help you

reduce the likelihood of over fitting after the tree is built. Of course, the pruning parameter is still used once we have built it.

max_depth

Limit the maximum depth of the tree, and all branches beyond the set depth will be cut off
This is the most widely used pruning parameter and is very effective in high dimension and low sample size. If the decision tree grows one more layer, the demand for sample size will be doubled, so limiting the depth of the tree can effectively limit over fitting. It is also very practical in the integration algorithm. In actual use, it is recommended to start with = 3 to see the effect of fitting and then decide whether to increase the set depth.

min_samples_leaf

min_ samples_ leaf defines that each child node of a node after branching must contain at least min_ samples_ leave training samples, otherwise the branching will not happen, or the branching will meet the requirement that each child node contains min_ samples_ Leave the direction of the sample to occur
General collocation Max_ Depth is used to make the model smoother and has magical effect in the regression tree. If the number of this parameter is set too small, it will cause over fitting, and if it is set too large, it will prevent the model from learning data. In general, it is recommended to start with = 5. If the sample size in the leaf node changes greatly, it is recommended to input floating-point numbers as the percentage of sample size. At the same time, this parameter can ensure the minimum size of each leaf and avoid the appearance of low variance and over fitting leaf nodes in regression problems. For classification problems with few categories, = 1 is usually the best choice.

min_samples_split

min_samples_split, a node must contain at least min_samples_split parts. Only when the training samples are split can the node be branched, otherwise the branching will not happen.

max_features

max_ features limit the number of features considered when branching, and features exceeding the limit will be discarded. And max_ depth, max_ features are pruning parameters used to limit the over fitting of high-dimensional data. However, the method is violent. It directly limits the number of features that can be used and forces the decision tree to stop. If the importance of each feature in the decision tree is not known, it may lead to insufficient model learning. If we want to prevent over fitting by dimension reduction, we suggest using PCA, ICA or dimension reduction algorithm in feature selection module.

min_impurity_decrease

min_impurity_decrease limits the size of information gain. Branches with information gain less than the set value will not occur.

class_weight

Parameters to complete sample label balance. Sample imbalance means that in a set of data sets, a class of tags naturally occupy a large proportion. For example, in the bank to judge "whether a

person who has a credit card will default" is the ratio of "no" (1%: 99%). In this classification situation, even if the model does nothing and predicts the result as "no", the accuracy rate can be 99%. So we're going to use class_ The weight parameter balances the sample labels, gives more weight to a small number of tags, and makes the model more inclined to a few classes and models towards the direction of capturing a few classes. This parameter defaults to none, which means that all labels in the dataset are automatically given the same weight.

min_weight_fraction_leaf

With the weight, the sample size is no longer simply the number of records, but affected by the weight of the input. Therefore, pruning at this time needs to match the min_ weight_ fraction_ Leaf is a pruning parameter based on weight. Also note the weight based pruning parameters (for example, min_ weight_ fraction_ Leaf) will be better than the criteria that do not know the sample weight (e.g. min_ samples_ Leaf) was less inclined to the dominant class. If the sample is weighted, the weight based pre pruning criterion is used to optimize the tree structure more easily, which ensures that the leaf node contains at least a small part of the sum of the sample weights.

Exercise 02:

base_ Estimator:

object or none, optional (default = none)
The basic estimator is suitable for the random subset of the data set. If it is none, the basic estimator is the decision tree.

n_observers:

int, optional (default is 10)
The basic estimator in the set.

max_samples:

int or float, optional (default = 1.0)
The number of samples extracted from X to train each basic estimator.If it is int, the sample Max is extracted_ samples. If float, this Max is extracted_ samples * X.shape[0]

max_features:

int or float, optional (default = 1.0)
The number of elements drawn from X to train each basic estimator.If int, the feature Max is drawn_ features. If it is floating, the feature Max is drawn_ features * X.shape[1]

bootstrap:

Boolean, optional (default = true)
Whether to take samples for replacement. If false, sampling without substitution is performed.

oob_score:

boolean variable, optional (default is false)

Whether to use ready-made samples to estimate the generalization error.

warm_start:

boolean variable, optional (default = false)
When set to true, reuse the solution from the last call to fit and add more estimators to the collection, otherwise, only for a brand new collection.

n_jobs:

int or none (optional) (default is none)
Fit and the number of jobs running in parallel predict. None unless joblib.parallel_ In the context of backend, otherwise it means 1. -1 means all processors are used.

random_state:

integer, randomstate instance or none, optional (default: none)
If int, then random_state is the seed used by the random number generator; otherwise, false.