# 决策树算法梳理

[决策树算法梳理 1](#_Toc16446663)

[1. 信息论基础（熵 联合熵 条件熵 信息增益 基尼不纯度） 1](#_Toc16446664)

[2. 决策树的不同分类算法（ID3算法、C4.5、CART分类树）的原理及应用场景 4](#_Toc16446665)

[3. 回归树原理 6](#_Toc16446666)

[4. 决策树防止过拟合手段 6](#_Toc16446667)

[5. 模型评估 9](#_Toc16446668)

[6. sklearn参数详解，Python绘制决策树 10](#_Toc16446669)

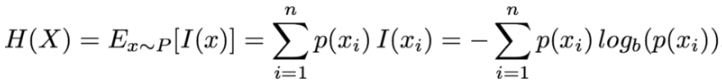
## 信息论基础（熵 联合熵 条件熵 信息增益 基尼不纯度）

#### 1）熵

在信息论与概率统计中，熵表示随机变量不确定性的度量。设X是一个取有限个值得离散随机变量，其概率分布为

P（X = xi ）= pi,i = 1,2, · · · ,n

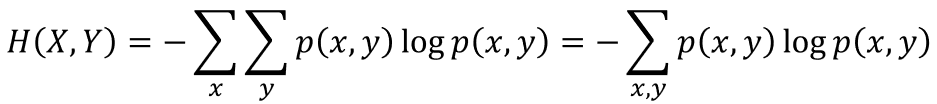
1则随机变量量X的熵定义为



若pi等于0，定义0log0 = 0，熵的单位为比特或者纳特

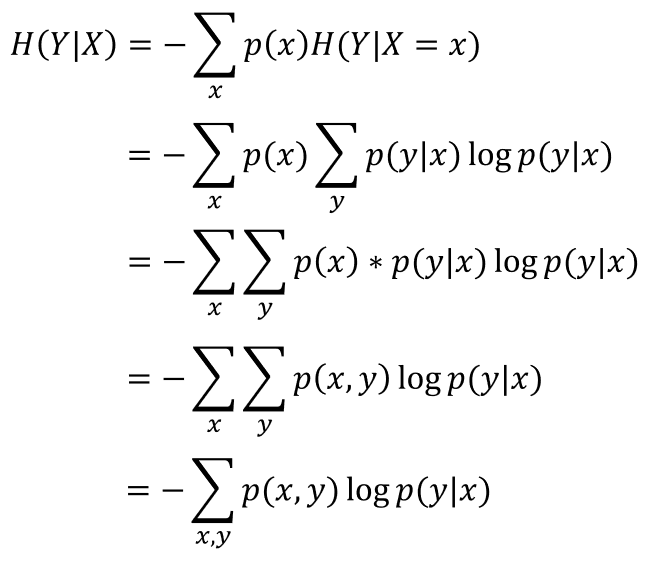
#### 2）联合熵

将一维随机变量分布推广到多维随机变量分布，则其**联合熵 (Joint entropy)**为：



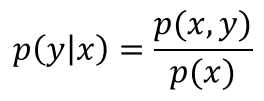
#### 3）条件熵

条件熵表示在已知事件x的条件下，事件y的不确定性。定义为**在给定条件下x，y的条件分布概率的熵对x的数学期望：**

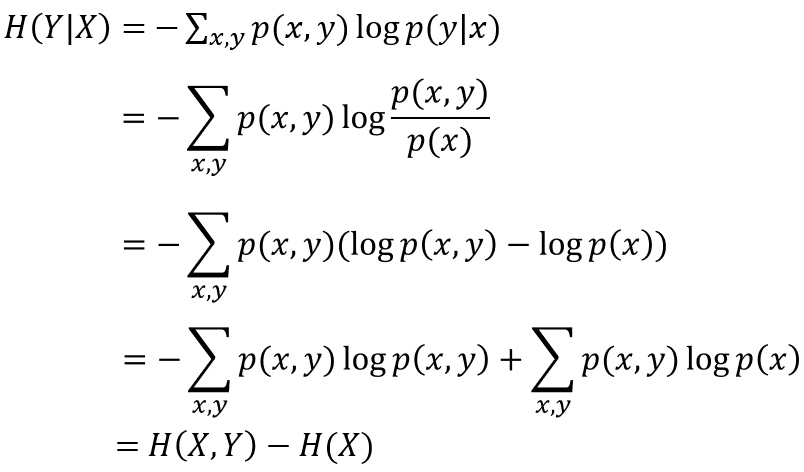


可以发现，**条件熵与联合熵仅仅在于log项不同。**

此外，根据联合概率分布与条件概率分布的关系，可得：



所以：



即在x条件下，y的条件熵 = x,y的联合熵 - x的信息熵

#### 4)信息增益

参考https://www.cnblogs.com/muzixi/p/6566803.html

**定义**： 以某特征划分数据集前后的熵的差值

      在熵的理解那部分提到了，熵可以表示样本集合的不确定性，熵越大，样本的不确定性就越大。因此可以**使用划分前后集合熵的差值来衡量使用当前特征对于样本集合D划分效果的好坏**。

      划分前样本集合D的熵是一定的 ，entroy(前)，

      使用某个特征A划分数据集D，计算划分后的数据子集的熵 entroy(后)

**信息增益 =** **entroy(前) -** **entroy(后)**

**书中公式:**

****

**做法：**计算使用所有特征划分数据集D，得到多个特征划分数据集D的信息增益，从这些信息增益中选择最大的，因而当前结点的划分特征便是使信息增益最大的划分所使用的特征。

**信息增益的理解：**

对于待划分的数据集D，其 entroy(前)是一定的，但是划分之后的熵 entroy(后)是不定的，entroy(后)越小说明使用此特征划分得到的子集的不确定性越小（也就是纯度越高），因此 entroy(前) -  entroy(后)差异越大，说明使用当前特征划分数据集D的话，其纯度上升的更快。而我们在构建最优的决策树的时候总希望能更快速到达纯度更高的集合，这一点可以参考优化算法中的梯度下降算法，每一步沿着负梯度方法最小化损失函数的原因就是负梯度方向是函数值减小最快的方向。同理：在决策树构建的过程中我们总是希望集合往最快到达纯度更高的子集合方向发展，因此我们总是选择使得信息增益最大的特征来划分当前数据集D。

**缺点**：信息增益偏向取值较多的特征

**原因**：当特征的取值较多时，根据此特征划分更容易得到纯度更高的子集，因此划分之后的熵更低，由于划分前的熵是一定的，因此信息增益更大，因此信息增益比较 偏向取值较多的特征。

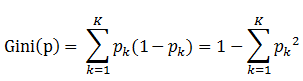
5）基尼不纯度

**定义：**基尼指数（基尼不纯度）：表示在样本集合中一个随机选中的样本被分错的概率。

**注意**： Gini指数越小表示集合中被选中的样本被分错的概率越小，也就是说集合的纯度越高，反之，集合越不纯。

即 **基尼指数（基尼不纯度）= 样本被选中的概率 \* 样本被分错的概率**

**书中公式：**



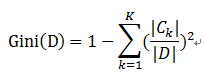
**说明:**

1. pk表示选中的样本属于k类别的概率，则这个样本被分错的概率是(1-pk)

2. 样本集合中有K个类别，一个随机选中的样本可以属于这k个类别中的任意一个，因而对类别就加和

3. 当为二分类是，Gini(P) = 2p(1-p)

**样本集合D的Gini指数 ：**假设集合中有K个类别，则：



## 决策树的不同分类算法（ID3算法、C4.5、CART分类树）的原理及应用场景

参考：https://shuwoom.com/?p=1452

#### ID3算法

输入：训练数据集D，特征集A，阈值ε；

输出：决策树T.

**Step1：**若D中所有实例属于同一类，则T为单结点树，并将类作为该节点的类标记，返回T；

**Step2：**若A=Ø，则T为单结点树，并将D中实例数最大的类作为该节点的类标记，返回T；

**Step3：**否则，2.1.1（3）计算A中个特征对D的信息增益，选择信息增益最大的特征；

**Step4：**如果的信息增益小于阈值ε，则T为单节点树，并将D中实例数最大的类作为该节点的类标记，返回T

**Step5：**否则，对的每一种可能值，依将D分割为若干非空子集，将中实例数最大的类作为标记，构建子结点，由结点及其子树构成树T，返回T；

**Step6：**对第i个子节点，以为训练集，以为特征集合，递归调用**Step1~step5**，得到子树，返回；

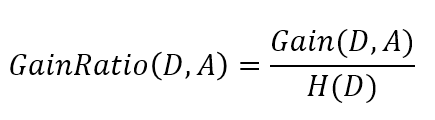
#### 2)C4.5

C4.5算法与ID3算法很相似，C4.5算法是对ID3算法做了改进，在生成决策树过程中采用信息增益比来选择特征。

信息增益比：

我们知道信息增益会偏向取值较多的特征，使用信息增益比可以对这一问题进行校正。

定义：特征A对训练数据集D的信息增益比GainRatio(D,A)定义为其信息增益Gain(D,A)与训练数据集D的经验熵H(D)之比：



#### 3)CART算法

输入：训练数据集D，停止计算的条件

输出：CART决策树

根据训练数据集，从根结点开始，递归地对每个结点进行以下操作，构建二叉树：

Step1：设结点的训练数据集为D，计算现有特征对该数据集的基尼指数。此时，对每一个特征A，对其可能取的每个值a，根据样本点A=a的测试为“是”或“否”将D分割为D1和D2两部分，利用上式Gini(D,A)来计算A=a时的基尼指数。

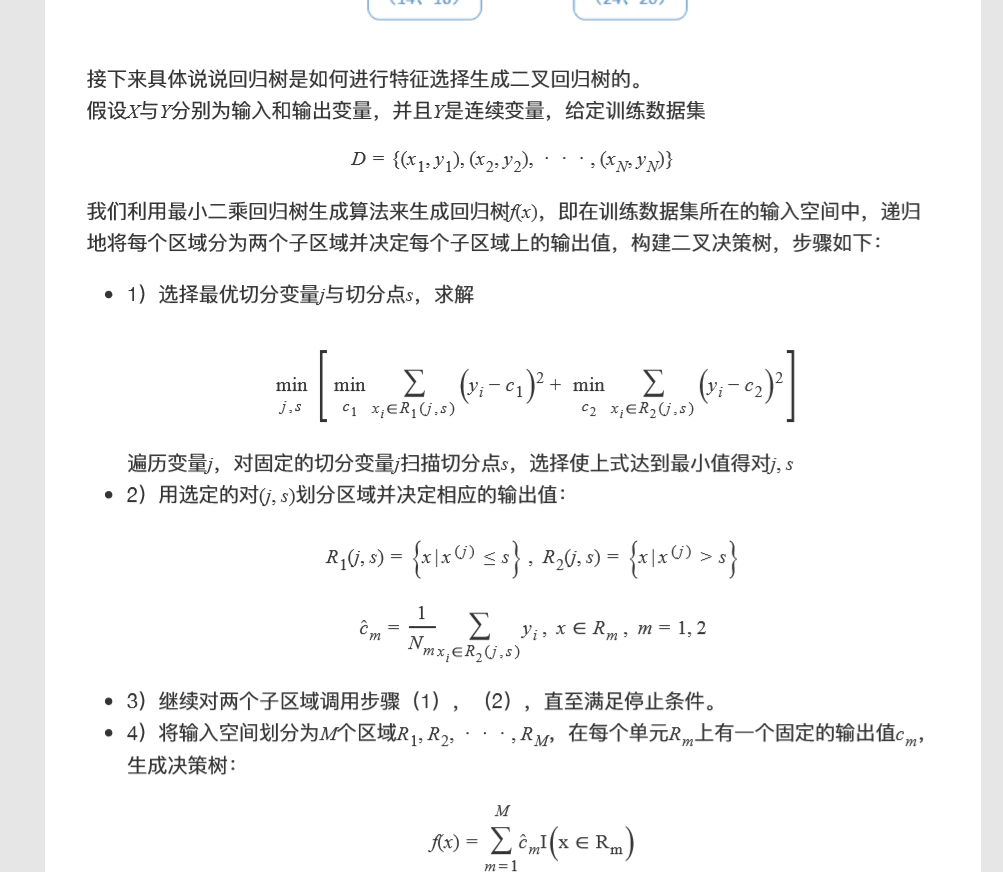
Step2：在所有可能的特征A以及他们所有可能的切分点a中，选择基尼指数最小的特征及其对应可能的切分点作为最有特征与最优切分点。依最优特征与最有切分点，从现结点生成两个子节点，将训练数据集依特征分配到两个子节点中去。

Step3：对两个子结点递归地调用Step1、Step2，直至满足条件。

Step4：生成CART决策树

算法停止计算的条件是节点中的样本个数小于预定阈值，或样本集的基尼指数小于预定阈值，或者没有更多特征

## 回归树原理



## 决策树防止过拟合手段

**剪枝类型：**预剪枝、后剪枝

* 预剪枝：在构造决策树的同时进行剪枝。所有决策树的构建方法，都是在无法进一步降低熵的情况下才会停止创建分支的过程，为了避免过拟合，可以设定一个阈值，熵减小的数量小于这个阈值，即使还可以继续降低熵，也停止继续创建分支。但是这种方法实际中的效果并不好。
* 后剪枝是在决策树生长完成之后，对树进行剪枝，得到简化版的决策树。

剪枝的过程是对拥有同样父节点的一组节点进行检查，判断如果将其合并，熵的增加量是否小于某一阈值。如果确实小，则这一组节点可以合并一个节点，其中包含了所有可能的结果。后剪枝是目前最普遍的做法。  
后剪枝的剪枝过程是删除一些子树，然后用其叶子节点代替，这个叶子节点所标识的类别通过大多数原则(majority class criterion)确定。所谓大多数原则，是指剪枝过程中, 将一些子树删除而用叶节点代替,这个叶节点所标识的类别用这棵子树中大多数训练样本所属的类别来标识,所标识的类称为majority class ，（majority class 在很多英文文献中也多次出现）。

**预剪枝依据：**

* 作为叶结点或作为根结点需要含的最少样本个数
* 决策树的层数
* 结点的经验熵小于某个阈值才停止

**后剪枝算法**

后剪枝算法有很多种，这里简要总结如下：

**Reduced-Error Pruning (REP,错误率降低剪枝）**

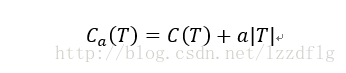
这个思路很直接，完全的决策树不是过度拟合么，我再搞一个测试数据集来纠正它。对于完全决策树中的每一个非叶子节点的子树，我们尝试着把它替换成一个叶子节点，该叶子节点的类别我们用子树所覆盖训练样本中存在最多的那个类来代替，这样就产生了一个简化决策树，然后比较这两个决策树在测试数据集中的表现，如果简化决策树在测试数据集中的错误比较少，那么该子树就可以替换成叶子节点。该算法以**bottom-up**的方式遍历所有的子树，直至没有任何子树可以替换使得测试数据集的表现得以改进时，算法就可以终止。

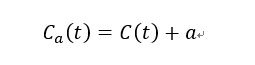
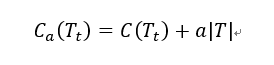
**Pessimistic Error Pruning (PEP，悲观剪枝）**

PEP剪枝算法是在C4.5决策树算法中提出的， 把一颗子树（具有多个叶子节点）用一个叶子节点来替代（我研究了很多文章貌似就是用子树的根来代替）的话，比起REP剪枝法，它不需要一个单独的测试数据集。

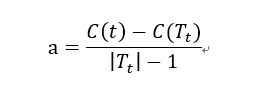
CART剪枝分为剪枝成子树序列，并通过交叉验证选取最优子树。

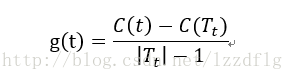
**1.剪枝，成子树序列**

在剪枝过程中，计算子树的损失函数：   
  
其中，**T为任意子树**，**C(T)为对训练数据的预测误差（如基尼指数）**，**|T|为子树的节点个数**，a>=0为参数，**Ca(T)为参数是a时子树T的整体损失**，**参数a权衡训练数据的拟合程度和模型的复杂度**。   
从上方a不是固定的时候，则整个损失函数由预测误差，a和模型复杂度共同决策，得出a大的时候，最优子树Ta偏小；当a小的时候，最优子树Ta的树较大，a=0时整体树最优，a->∞时，根结点组成的单节点树最优。   
**且认为不同的a值，可以确定一棵不同的最优化的树**，或者说一个区间内的a值可以确定一颗最优化的树   
即将a不断增大，利用a生成Ta这棵最优子树   
这里写图片描述

为了得到**所有的可能生成的最优化树**{T0,T1,T2,…Tn},我们**须从底向上，每次进行一次剪枝，通过得到的树认为是最优化树反推a**   
**具体的，从整体树T0开始剪枝，对于T0的任意内部结点t，结点下有若干子节点，把t下的子树的若干叶节点称为Tt。**   
剪枝后(**即t下的子树减去后，t变为叶节点**)的损失函数   
  
剪枝前的损失函数   
  
注意这里的损失函数都是某个叶结点的损失函数，但为什么可以只求叶节点的损失函数，因为在上面的分析ID3,C4.5中得出了C(T)作为模型的预测误差的值，通过累加每一个叶结点（即T个叶结点）的预测误差而得出C(T)。因此单独求某个叶结点并没有什么问题。

现在就是求解a，a如何求解？   
当a=0或充分小时，不等式

这里写图片描述  
因为叶结点越多预测误差应该越小。   
当a不断增大时，在某个a点有   
这里写图片描述  
当a再增大时，不等式反向。因此只要   
  
Tt与t有相同的损失函数，而t结点少，因此t比Tt更可取，对Tt进行剪枝。   
接下来对T0这棵整体树中的每一个结点t，计算



这个g(t)表示剪枝后的整体损失函数减少程度，实际上可以看为是否剪枝的阈值，对于某个结点当他的参数a=g(t)时，剪和不剪总体损失函数时一样的。如果a增大则不剪的整体损失函数就大于剪去的。即a大于g(t)该剪，剪后会使整体损失函数减小，而a小于g(t)则不剪，剪后会使整体损失函数增大。   
**这样a缓慢增大，随着a的增大，在一个区间内可确定一棵最优的剪枝树**

**而我们求每棵树，并认为他是最优剪枝树。**

**g(t)则代表每一棵树的a的最优区间内的最小值**

即在T0中剪去g(t)最小的Tt,得到的子树为T1,同时将最小的g(t)设为a1,那

么T1为区间[a1,a2)的最优子树。

如此这样下去，将所有可能的树的情况剪枝直到根节点，在这个过程中则

会不断增加a，产生新的区间，最后a的所有可能的g(t)取值全部确定。

**2.通过交叉验证选取最优子树**

具体的利用独立的验证数据集，测试子树序列T0,T1,…Tn中各棵子树的平

方误差或基尼指数。平方误差或基尼指数最小的决策数被认为是最优决策

数，因为我们每确定一棵子树就会确定其参数a值，所以最优子树Tk确定，对应ak也确定，即得到最优决策数Ta

## 模型评估

1.构建分类模型的非参数方法  
2.NP完全问题  
3.计算代价小  
4.决策树容易解释  
5.学习离散值  
6.避免过分拟合  
7.冗余数据不会造成影响  
8.叶结点记录少，不具统计意义：数据碎片问题设置阈值  
9.子树重复问题  
10.测试条件只涉及一个属性：斜决策树  
11.不纯度量方法影响小

## 6. sklearn参数详解，Python绘制决策树

参考：https://blog.csdn.net/qq\_16000815/article/details/80954039

scikit-learn中有两类决策树，它们均采用优化的CART决策树算法。

'''

from sklearn.tree import DecisionTreeRegressor

'''

回归决策树

'''

DecisionTreeRegressor(criterion="mse",

splitter="best",

max\_depth=None,

min\_samples\_split=2,

min\_samples\_leaf=1,

min\_weight\_fraction\_leaf=0.,

max\_features=None,

random\_state=None,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.,

min\_impurity\_split=None,

presort=False)

'''

参数含义：

1.criterion:string, optional (default="mse")

它指定了切分质量的评价准则。默认为'mse'(mean squared error)。

2.splitter:string, optional (default="best")

它指定了在每个节点切分的策略。有两种切分策咯：

(1).splitter='best':表示选择最优的切分特征和切分点。

(2).splitter='random':表示随机切分。

3.max\_depth:int or None, optional (default=None)

指定树的最大深度。如果为None，则表示树的深度不限，直到

每个叶子都是纯净的，即叶节点中所有样本都属于同一个类别，

或者叶子节点中包含小于min\_samples\_split个样本。

4.min\_samples\_split:int, float, optional (default=2)

整数或者浮点数，默认为2。它指定了分裂一个内部节点(非叶子节点)

需要的最小样本数。如果为浮点数(0到1之间)，最少样本分割数为ceil(min\_samples\_split \* n\_samples)

5.min\_samples\_leaf:int, float, optional (default=1)

整数或者浮点数，默认为1。它指定了每个叶子节点包含的最少样本数。

如果为浮点数(0到1之间)，每个叶子节点包含的最少样本数为ceil(min\_samples\_leaf \* n\_samples)

6.min\_weight\_fraction\_leaf:float, optional (default=0.)

它指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

7.max\_feature:int, float, string or None, optional (default=None)

可以是整数，浮点数，字符串或者None。默认为None。

(1).如果是整数，则每次节点分裂只考虑max\_feature个特征。

(2).如果是浮点数(0到1之间)，则每次分裂节点的时候只考虑int(max\_features \* n\_features)个特征。

(3).如果是字符串'auto',max\_features=n\_features。

(4).如果是字符串'sqrt',max\_features=sqrt(n\_features)。

(5).如果是字符串'log2',max\_features=log2(n\_features)。

(6).如果是None，max\_feature=n\_feature。

8.random\_state:int, RandomState instance or None, optional (default=None)

(1).如果为整数，则它指定了随机数生成器的种子。

(2).如果为RandomState实例，则指定了随机数生成器。

(3).如果为None，则使用默认的随机数生成器。

9.max\_leaf\_nodes:int or None, optional (default=None)

(1).如果为None，则叶子节点数量不限。

(2).如果不为None，则max\_depth被忽略。

10.min\_impurity\_decrease:float, optional (default=0.)

如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)大于或等于min\_impurity\_decrease，则分裂该节点。

个人理解这个参数应该是针对分类问题时才有意义。这里的不纯度应该是指基尼指数。

回归生成树采用的是平方误差最小化策略。分类生成树采用的是基尼指数最小化策略。

加权不纯度的减少量计算公式为：

min\_impurity\_decrease=N\_t / N \* (impurity - N\_t\_R / N\_t \* right\_impurity

- N\_t\_L / N\_t \* left\_impurity)

其中N是样本的总数，N\_t是当前节点的样本数，N\_t\_L是分裂后左子节点的样本数，

N\_t\_R是分裂后右子节点的样本数。impurity指当前节点的基尼指数，right\_impurity指

分裂后右子节点的基尼指数。left\_impurity指分裂后左子节点的基尼指数。

11.min\_impurity\_split:float

树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节点将分裂，否则它是叶子节点。

这个参数已经被弃用。用min\_impurity\_decrease代替了min\_impurity\_split。

12.presort： bool, optional (default=False)

指定是否需要提前排序数据从而加速寻找最优切分的过程。设置为True时，对于大数据集

会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深度的情况下，会加速训练过程。

属性：

1.feature\_importances\_ : array of shape = [n\_features]

特征重要性。该值越高，该特征越重要。

特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也被称为基尼的重要性

2.max\_feature\_:int

max\_features推断值。

3.n\_features\_：int

执行fit的时候，特征的数量。

4.n\_outputs\_ : int

执行fit的时候，输出的数量。

5.tree\_ : 底层的Tree对象。

Notes：

控制树大小的参数的默认值（例如``max\_depth``，``min\_samples\_leaf``等）导致完全成长和未剪枝的树，

这些树在某些数据集上可能表现很好。为减少内存消耗，应通过设置这些参数值来控制树的复杂度和大小。

方法：

1.fit(X,y):训练模型。

2.predict(X):预测。

'''

from sklearn.tree import DecisionTreeClassifier

'''

分类决策树

'''

DecisionTreeClassifier(criterion="gini",

splitter="best",

max\_depth=None,

min\_samples\_split=2,

min\_samples\_leaf=1,

min\_weight\_fraction\_leaf=0.,

max\_features=None,

random\_state=None,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.,

min\_impurity\_split=None,

class\_weight=None,

presort=False)

'''

参数含义：

1.criterion:string, optional (default="gini")

(1).criterion='gini',分裂节点时评价准则是Gini指数。

(2).criterion='entropy',分裂节点时的评价指标是信息增益。

2.max\_depth:int or None, optional (default=None)。指定树的最大深度。

如果为None，表示树的深度不限。直到所有的叶子节点都是纯净的，即叶子节点

中所有的样本点都属于同一个类别。或者每个叶子节点包含的样本数小于min\_samples\_split。

3.splitter:string, optional (default="best")。指定分裂节点时的策略。

(1).splitter='best',表示选择最优的分裂策略。

(2).splitter='random',表示选择最好的随机切分策略。

4.min\_samples\_split:int, float, optional (default=2)。表示分裂一个内部节点需要的做少样本数。

(1).如果为整数，则min\_samples\_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每次分裂最少样本数为ceil(min\_samples\_split \* n\_samples)

5.min\_samples\_leaf: int, float, optional (default=1)。指定每个叶子节点需要的最少样本数。

(1).如果为整数，则min\_samples\_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每个叶子节点最少样本数为ceil(min\_samples\_leaf \* n\_samples)

6.min\_weight\_fraction\_leaf:float, optional (default=0.)

指定叶子节点中样本的最小权重。

7.max\_features:int, float, string or None, optional (default=None).

搜寻最佳划分的时候考虑的特征数量。

(1).如果为整数，每次分裂只考虑max\_features个特征。

(2).如果为浮点数(0到1之间)，每次切分只考虑int(max\_features \* n\_features)个特征。

(3).如果为'auto'或者'sqrt',则每次切分只考虑sqrt(n\_features)个特征

(4).如果为'log2',则每次切分只考虑log2(n\_features)个特征。

(5).如果为None,则每次切分考虑n\_features个特征。

(6).如果已经考虑了max\_features个特征，但还是没有找到一个有效的切分，那么还会继续寻找

下一个特征，直到找到一个有效的切分为止。

8.random\_state:int, RandomState instance or None, optional (default=None)

(1).如果为整数，则它指定了随机数生成器的种子。

(2).如果为RandomState实例，则指定了随机数生成器。

(3).如果为None，则使用默认的随机数生成器。

9.max\_leaf\_nodes: int or None, optional (default=None)。指定了叶子节点的最大数量。

(1).如果为None,叶子节点数量不限。

(2).如果为整数，则max\_depth被忽略。

10.min\_impurity\_decrease:float, optional (default=0.)

如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)大于或等于min\_impurity\_decrease，则分裂该节点。

加权不纯度的减少量计算公式为：

min\_impurity\_decrease=N\_t / N \* (impurity - N\_t\_R / N\_t \* right\_impurity

- N\_t\_L / N\_t \* left\_impurity)

其中N是样本的总数，N\_t是当前节点的样本数，N\_t\_L是分裂后左子节点的样本数，

N\_t\_R是分裂后右子节点的样本数。impurity指当前节点的基尼指数，right\_impurity指

分裂后右子节点的基尼指数。left\_impurity指分裂后左子节点的基尼指数。

11.min\_impurity\_split:float

树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节点将分裂，否则它是叶子节点。

这个参数已经被弃用。用min\_impurity\_decrease代替了min\_impurity\_split。

12.class\_weight:dict, list of dicts, "balanced" or None, default=None

类别权重的形式为{class\_label: weight}

(1).如果没有给出每个类别的权重，则每个类别的权重都为1。

(2).如果class\_weight='balanced'，则分类的权重与样本中每个类别出现的频率成反比。

计算公式为：n\_samples / (n\_classes \* np.bincount(y))

(3).如果sample\_weight提供了样本权重(由fit方法提供)，则这些权重都会乘以sample\_weight。

13.presort:bool, optional (default=False)

指定是否需要提前排序数据从而加速训练中寻找最优切分的过程。设置为True时，对于大数据集

会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深度的情况下，会加速训练过程。

属性:

1.classes\_:array of shape = [n\_classes] or a list of such arrays

类别的标签值。

2.feature\_importances\_ : array of shape = [n\_features]

特征重要性。越高，特征越重要。

特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也被称为基尼的重要性

3.max\_features\_ : int

max\_features的推断值。

4.n\_classes\_ : int or list

类别的数量

5.n\_features\_ : int

执行fit后，特征的数量

6.n\_outputs\_ : int

执行fit后，输出的数量

7.tree\_ : Tree object

树对象，即底层的决策树。

方法:

1.fit(X,y):训练模型。

2.predict(X):预测

3.predict\_log\_poba(X):预测X为各个类别的概率对数值。

4.predict\_proba(X):预测X为各个类别的概率值。