目录

[**决策树** 1](#_Toc532315556)

[**一、** **概述** 1](#_Toc532315557)

[**二、** **步骤** 2](#_Toc532315558)

[**三、** **特征选择** 2](#_Toc532315559)

[3．1信息熵（Entropy） 3](#_Toc532315560)

[3．2条件熵（Conditional Entropy） 3](#_Toc532315561)

[3．3信息增益（Information Gain） 4](#_Toc532315562)

[3．4信息增益比(Gain ratio) 6](#_Toc532315563)

[3．5基尼指数（Gini index） 6](#_Toc532315564)

[**四、** **剪枝** 7](#_Toc532315565)

[4．1预备信息 8](#_Toc532315566)

[4．2预剪枝 9](#_Toc532315567)

[4．3后剪枝 10](#_Toc532315568)

[**五、** **代码** 11](#_Toc532315569)

1. **概述**

决策树（Decision Tree）又称为判定树，是运用于分类的一种树结构，其中的每个内部节点代表对某一属性的一次测试，每条边代表一个测试结果，叶节点代表某个类或类的分布。

决策树的决策过程需要从决策树的根节点开始，待测数据与决策树中的特征节点进行比较，并按照比较结果选择选择下一比较分支，直到叶子节点作为最终的决策结果。

相亲对象分类系统:

选择对象

是否有房

值得考虑

是否上进

否

是

拜拜

备胎

否

是

1. **步骤**
2. 特征选择
3. 决策树生成
4. 剪枝
5. **特征选择**



**放贷数据**

## 3．1信息熵（Entropy）

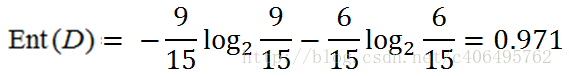
熵定义为信息的期望值。在信息论与概率统计中，熵是表示随机变量不确定性的度量。

假定当前样本集合D中第k类样本所占比例为𝑝\_𝑘 （𝑘=1,2,…,|y|）,则D的信息熵定义为



其中，|y|是分类的数目。End(D)的值越大，则D的不确定性越高。

以放贷数据D为例，15条数据中，9个放贷，6个不放贷，则该数据集D的信息熵：

****

## 3．2条件熵（Conditional Entropy）

条件熵Ent(D|a)表示在已知随机变量a的条件下随机变量D的不确定性，公式如下：

****

其中，离散属性a有V个可能的取值{𝑎^1, 𝑎^2,…., 𝑎^𝑉}，若使用a来对样本集D进行划分，则会产生V个分支结点，其中第v个分支结点包含了D中所有在属性a上取值为𝑎^𝑣的样本，记为𝐷^𝑣。计算出𝐷^𝑣的信息熵，再考虑到不同的分支结点所包含的样本数不同，给分支结点赋予权重|𝐷^𝑣 |/|𝐷| ，即样本数越多的分支结点的影响越大。

## 3．3信息增益（Information Gain）

如何选择特征，需要看信息增益。

也就是说，信息增益是相对于特征而言的，信息增益越大，特征对最终的分类结果影响也就越大。

特征a对训练数据集D的信息增益Gain(D,a)，定义为集合D的信息熵Ent(D)与属性a给定条件下D的条件熵Ent(D|a)之差，即



以贷款申请样本数据表为例：

看下年龄这一列的数据，也就是特征a1，一共有三个类别，分别是：青年、中年和老年。

年龄是青年的数据一共有5个，

所以P（年龄=青年）=5/15=1/3，P（年龄=中年）=5/15=1/3，P（年龄=老年）=5/15=1/3

P（贷款|年龄=青年）=2/5 P（贷款|年龄=中年）=3/5 P（贷款|年龄=老年）=4/5

所以年龄的信息增益：



同理，计算其余特征的信息增益Gain(D,a2)、 Gain(D,a3) 和Gain(D,a4)。分别为：





最后，比较特征的信息增益，由于特征a3(有自己的房子)的信息增益值最大，所以选择a3作为最优特征。

基于“有自己的房子”对根节点进行划分的结果，有自己的房子都放贷，没有自己的房子的部分放贷，部分不放贷，这部分数据需要继续分支。

以第二个分支结点（“有自己的房子=否”）为例：

该结点包含的样例集合D1，中有编号为{1,2,3,5,6,7,13,14,15}的9个样例，可用属性集合为{年龄，有工作，信贷情况}。

该结点的信息熵为：



基于D1计算出各属性的信息增益：

P（年龄=青年）=4/9， P（ 贷款|年龄=青年）=1/4，

P（年龄=中年）=2/9， P（ 贷款|年龄=中年）=0，

P（年龄=老年）=3/9=1/3。 P（ 贷款|年龄=老年）=2/3。



有自己的房子

4,8,9,10,11,12

1,2,3,5,6,7,13,14,15

是

否

放贷

放贷

不放贷

对于待划分的数据集D，其 Entropy (前)是一定的，但是划分之后的熵 Entropy (后)是不定的， Entropy (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 Entropy (前) - Entropy (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。

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，例如放贷数据中的ID，以ID划分将产生15个分支，每个分支结点仅包含一个样本，这些分支结点的纯度已达到最大。但是，明显这样的决策树是没有意义的，即不具有泛化性。

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。

## 3．4信息增益比(Gain ratio)

对于由于信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好带来的不利影响，著名的C4.5决策树算法不直接使用信息增益，而是使用 “信息增益比”(gain ratio)来选择最优划分属性。

信息增益比定义为：



其中，



称为属性a的“固有值”。属性a的可能取值数目越多即V越大，则IV(a)的值通常越大。

所以信息增益比准则对可取值数目较少的属性有所偏好，因此，C4.5算法并不是直接选择增益比最大的候选划分属性，而是使用了一个启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益比最高的。

## 3．5基尼指数（Gini index）

CART决策树使用“基尼指数”（Gini index）来选择划分属性。

数据D的纯度可用基尼值来度量：



直观来讲，Gini(D)反映了从数据集D中随机抽取两个样本，其类别标记不一致的概率。

因此，Gini(D)越小，数据集D的不确定性越小，纯度越高。

基尼指数定义为：



最终选择基尼指数最小的属性作为最优划分属性。

1. **剪枝**

剪枝(pruning)是决策树学习算法对付“过拟合”的主要手段。

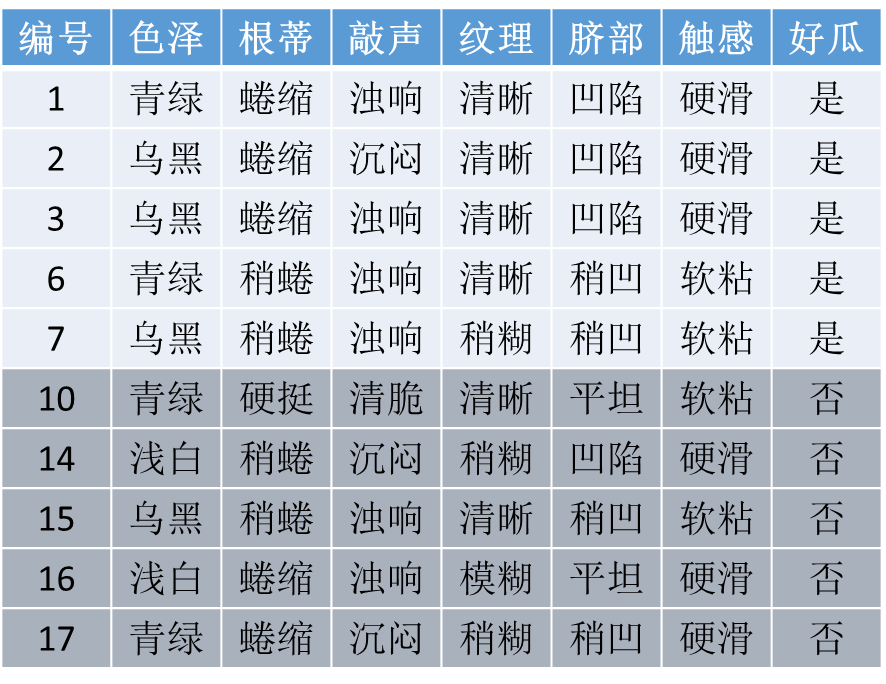
在决策树学习中，为了尽可能正确分类训练样本，结点划分过程将不断重复，有时会导致决策树分支过多，这时就可能因训练样本学得“太好”了，以至于把训练集自身的一些特点当做所有数据都具有的一般性质而导致过拟合。

因此，可通过主动去掉一些分支来降低过拟合的风险。

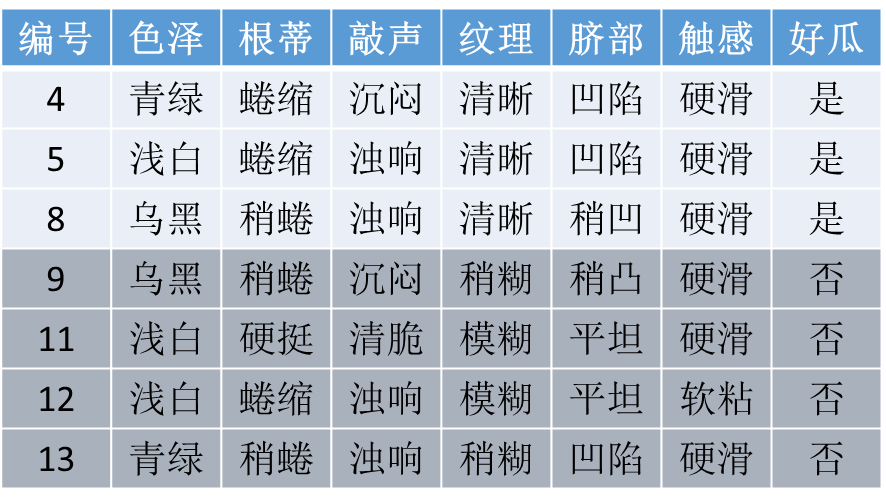
决策树剪枝的基本策略有“预剪枝”(prepruning)和“后剪枝”(postpruning)。

本将以西瓜数据集（训练集和验证集）为例。

训练集：



验证集：



## 4．1预备信息

### 4．1．1错误率/精度

CART分类错误的样本数量占样本总数的比例称为“错误率”(error rate)，

即如果在m个样本中有a个样本分类错误，则错误率：E=a/m

相应的，精度(accuracy)=1-错误率。

### 4．1．2误差

学习器的实际预测输出与样本的真实输出之间的差异称为“误差”(error)，

学习器在训练集上的误差称为“训练误差”(training error)或“经验误差”(empirical error)，

在新样本上的误差称为“泛化误差”(generalization error)。

### 4．1．3过拟合

当学习器把训练样本学得“太好”了的时候，很可能已经把训练样本自身的一些特点当做了所有潜在样本都会有的一般性质，这样会导致泛化性能下降，这种现象就称为“过拟合”(overfitting)。

## 4．2预剪枝

定义：对每个结点在划分前后进行估计，若当前结点的划分不能带来决策树泛化性能提升，则停止划分并将当前结点标记为叶节点

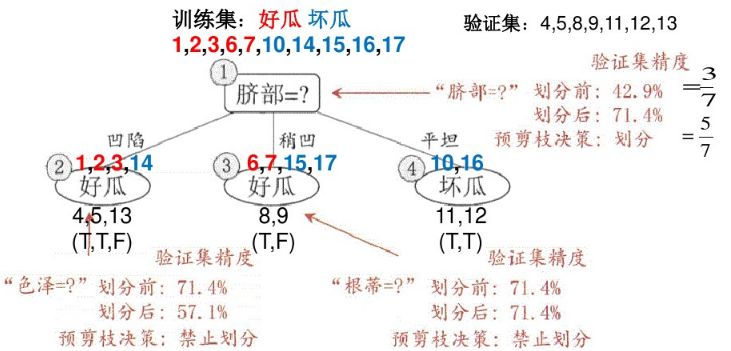
预剪枝基于信息增益准则，选择属性“脐部”来对训练集进行划分，并产生3个分支。预剪枝要对划分前后的泛化性能进行评估。

在划分前，所有的样例集中在根结点。

►若不进行划分，该结点将被标记为叶结点，其类别标记为训练样例最多的类别，由于训练集中正反例数量相等，假定这个叶结点标记为“好瓜”。用验证集对这个单节点决策树进行评估，则编号为{4,5,8}的样例被分类正确，另外4个样例分类错误，于是验证集精度为3/7=42.9%

►若基于“脐部”进行划分后，如右图所示，三个结点分别标记为“好瓜”、“好瓜”、“坏瓜”。此时验证集精度为5/7=71.4%>42.9%。

于是，在基于属性“脐部”划分后，泛化性能得到提升。



接着，决策树算法应该对结点2进行划分，基于信息增益准则将挑选出划分属性“色泽”。

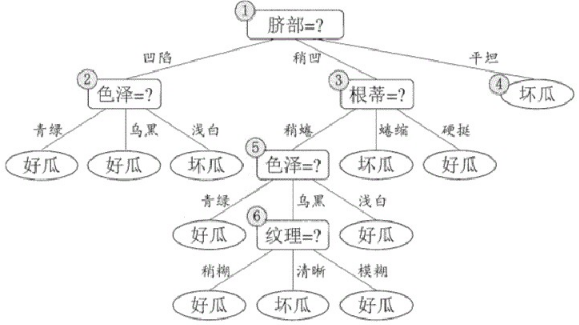
同理，计算划分前后的验证集精度，发现根据属性“色泽”划分后验证集精度下降为57.1%。

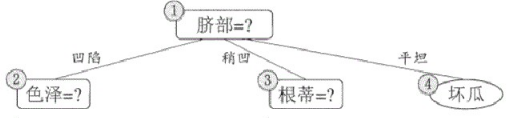
于是，预剪枝策略将禁止结点2被划分。

对于结点3，划分后验证集精度与划分前一样，说明这个划分并不能提升验证集精度，故禁止结点3被划分。

对于结点4，所含训练集已属于同一类，不再进行划分。

故最终经过预剪枝的决策树生成，其验证集精度为71.4%，并且泛化能力得到了提高。





优点：降低了过拟合的风险，减少了决策树的训练时间开销和测试时间开销。

缺点：基于“贪心”本质禁止部分分支展开，会带来欠拟合的风险。

## 4．3后剪枝

定义：先从训练集生成一颗完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点。

后剪枝先从训练集生成一颗完整决策树，如下左图，其验证集精度为42.9%。

后剪枝先考察结点6，若将结点6的分支剪除，则相当于结点6替换为叶结点。替换后的叶结点包含训练样本{7,15}。

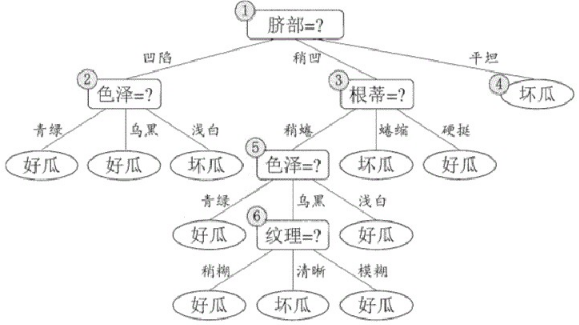
故该分支标记为“好瓜”，此时决策树的验证集精度提高至57.1%，于是，可以不剪枝。

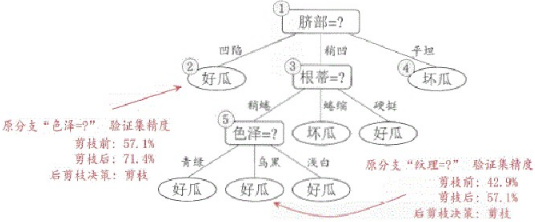
同理，考察结点5，训练样例{6,7,15}，标记为“好瓜”，精度不提高，可以不进行剪枝。

对于结点2，训练样例{1,2,3,14}，标记为“好瓜”，精度提高至71.4%，故决定剪枝。

对于结点3和1，泛化性能均未得到提高，故保留。

最终经过后剪枝得到的决策树，其验证集精度为71.4%，泛化性能得到提升。



‘

1. **代码**

## 5．1 Python函数

参考：<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

<https://www.cnblogs.com/pinard/p/6056319.html>

scikit-learn决策树算法类库内部实现是使用了调优过的CART树算法。

classsklearn.tree.**DecisionTreeClassifier**(criterion=’gini’, splitter=’best’, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, class\_weight=None, presort=False)[[source]](https://github.com/scikit-learn/scikit-learn/blob/55bf5d9/sklearn/tree/tree.py#L519)

参数：

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

**默认使用的划分准则是基尼指数，criterion参数支持基尼指数(criterion=’’ gini”)和信息增益(criterion=’’ entropy”)**

1. **splitter : *string, optional (default=”best”)***

**在构造树时，选择结点的原则，默认是选择最好的特征点分类（splitter='best'），比如基于信息增益分类时，则选择信息增益最大的特征点。还可以是随机的在部分特征点中找局部最优的特征点(splitter=”** random**”)。**

**默认的"best"适合样本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐"random"**

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

**表示树的最大深度。默认是"None",意味着决策树在建立子树的时候不会限制子树的深度。**

**一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。**

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

**内部节点再划分所需最小样本数。**

**这个值限制子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。默认是2。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。  
 1).如果是int，将其最为最小的样本数。  
 2)如果是float，min\_samples\_split是一个百分率，并且ceil(min\_samples\_split**

**\*n\_samples)是每个分类需要的样本数。ceil是取大于或等于指定表达式的最小整数。**

**我之前一个项目，大概10万样本，建立决策树时，设定min\_samples\_split=10。可以作为参考。**

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

这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。

1.如果是int，则其为最小样本数

2.如果是float，则它是一个百分率并且ceil(min\_samples\_leaf\*n\_samples)是每个节点所需的样本数。

如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

之前的10万样本项目使用min\_samples\_leaf的值为5，仅供参考。

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

这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。

默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

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

在进行分类时需要考虑的特征数。

1).如果是int，在每次分类是都要考虑max\_features个特征。

2).如果是float,那么max\_features是一个百分率并且分类时需要考虑的特征数是int(max\_features\*n\_features,其中n\_features是训练完成时的特征数)。

3).如果是auto,max\_features=sqrt(n\_features)

4).如果是sqrt,max\_features=sqrt(n\_features)

5).如果是log2,max\_features=log2(n\_features)

6).如果是None，max\_features=n\_features

注意：至少找到一个样本点有效地被分类时，搜索分类才会停止。

一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

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

如果是int,random\_state 是随机数字发生器的种子；如果是RandomState，random\_state是随机数字发生器，如果是None，随机数字发生器是np.random使用的RandomState instance.

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

通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。

如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

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

如果这种分裂导致不确定性的减少大于或等于这个值，节点将被分裂。

1. **min\_impurity\_split : *float, (default=1e-7)***

这个值限制了决策树的增长，如果某节点的不确定性(基尼系数，信息增益，均方差，绝对差)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。

1. **class\_weight : *dict, list of dicts, “balanced” or None, default=None***

指定样本各类别的的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别。这里可以自己指定各个样本的权重，或者用“balanced”，如果使用“balanced”，则算法会自己计算权重，样本量少的类别所对应的样本权重会高。当然，如果你的样本类别分布没有明显的偏倚，则可以不管这个参数，选择默认的"None"。

表示在表{class\_label:weight}中的类的关联权值。如果没有指定，所有类的权值都为1。

如果没有给出，所有类的权值都应该是1。对于多输出问题，可以按照与y列相同的顺序提供一个dicts列表。

请注意，对于多输出（包括多标记），应为其自己的dict中的每个列的每个类定义权重。

例如，对于四类多标签分类权重应为[{0：1,1：1}，{0：1,1：5}，{0：1,1：1}，{0：1,1： 1}]而不是[{1：1}，{2：5}，{3：1}，{4：1}]。

"balanced"模型使用y的值去自动适应权值，并且是以输入数据中类的频率的反比例。如：n\_samples/(n\_classes\*np.bincount(y))。

对于多输出，每列y的权值都会相乘。

如果sample\_weight已经指定了，这些权值将与samples以合适的方法相乘。

1. **presort : *bool, optional (default=False)***

数据是否预排序。

这个值是布尔值，默认是False不排序。一般来说，如果样本量少或者限制了一个深度很小的决策树，设置为true可以让划分点选择更加快，决策树建立的更加快。如果样本量太大的话，反而没有什么好处。问题是样本量少的时候，我速度本来就不慢。所以这个值一般懒得理它就可以了。

属性：

1. **classes\_ : array of shape = [n\_classes] or a list of such arrays**  
   类标签（单输出问题），或类标签数组列表（多输出问题）。
2. [**feature\_importances\_**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.feature_importances_)**: array of shape = [n\_features]**

返回要素重要性。

1. **max\_features\_ : *int,***

max\_features的推断值。

1. **n\_classes\_ : *int or list***

类的数量（对于单个输出问题），或包含每个输出的类数的列表（对于多输出问题）。

1. **n\_features\_ : *int***

当fit被执行的时候，特征的数量。

1. **n\_outputs\_ : *int***

当fit被执行的时候，输出的数量。

1. **tree\_ : *Tree object***

底层的Tree对象。

注意：

默认参数控制树的大小（例如：max\_depth, min\_samples\_leaf,等）导致树的完全成长和未修剪，这些树在某些数据集上可能非常大。为减少内存消耗，应通过设置这些参数值来控制树的复杂性和大小。

每次拆分时，这些特征总是随机置换。因此，即使使用相同的训练数据和max\_features = n\_features，如果在搜索最佳分割期间枚举的几个分割的标准的改进相同，则找到的最佳分割会变化。

为了在fitting期间获得准确的答案，需要修改random\_state。

1）当样本数量少但样本特征非常多的时候，决策树很容易过拟合，一般来说，样本数比特征数多一些会比较容易建立健壮的模型。

2）如果样本数量少但样本特征非常多，在拟合决策树模型前，推荐先做维度规约，比如主成分分析（PCA），特征选择（Losso）或者独立成分分析（ICA）。这样特征的维度会大大减小。再来拟合决策树模型效果会好。

3）推荐多用决策树的可视化（下节会讲），同时先限制决策树的深度（比如最多3层），这样可以先观察下生成的决策树里数据的初步拟合情况，然后再决定是否要增加深度。

4）在训练模型先，注意观察样本的类别情况（主要指分类树），如果类别分布非常不均匀，就要考虑用class\_weight来限制模型过于偏向样本多的类别。

5）决策树的数组使用的是numpy的float32类型，如果训练数据不是这样的格式，算法会先做copy再运行。

6）如果输入的样本矩阵是稀疏的，推荐在拟合前调用csc\_matrix稀疏化，在预测前调用csr\_matrix稀疏化。

方法：

1. [**apply**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.apply)(X[, check\_input=True])  
   返回每个样本被预测为的叶子的索引。

**参数：**

1）**X : *array\_like or sparse matrix, shape = [n\_samples, n\_features]、***

输入样本。在内部，它将被转换为dtype=np.float32，并且如果向稀疏csr\_matrix提供稀疏矩阵。

2）**check\_input : *boolean, (default=True)***

允许绕过多个输入检查。除非您知道自己的操作，否则请勿使用此参数。

**返回：**

**X\_leaves : *array\_like, shape = [n\_samples,]***

对于X中的每个数据点x，返回叶子x的最终索引。叶子以[0;self.tree\_.node\_count）形式编号，可能与编号有差距。

1. [**decision\_path**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.decision_path)(X[, check\_input])

返回树中的决策路径。

**参数：**

1）**X : *array\_like or sparse matrix, shape = [n\_samples, n\_features]***

输入样本。在内部，它将被转换为dtype=np.float32，并且如果向稀疏csr\_matrix提供稀疏矩阵。

2）**check\_input : *boolean, (default=True)***

允许绕过多个输入检查。除非您知道自己的操作，否则请勿使用此参数。

**返回：**

**indicator : *sparse csr array, shape = [n\_samples, n\_nodes]***

返回节点指示符矩阵，其中非零元素指示样本通过节点。

1. [**fit**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.fit)(X, y[, sample\_weight, check\_input, …])

从训练集（X，y）构建决策树分类器。

**参数：**

1）**X : *array-like or sparse matrix, shape = [n\_samples, n\_features]***

输入样本。在内部，它将被转换为dtype=np.float32，并且如果向稀疏csr\_matrix提供稀疏矩阵。

2）**y : *array-like, shape = [n\_samples] or [n\_samples, n\_outputs]***

目标值（类标签）是整形或者字符串。

3）**sample\_weight : *array-like, shape = [n\_samples] or None***

样本权重。如果为None，则样本的权重相等。在每个节点中搜索拆分且忽略具有净零或负权重的子节点将创建子节点。如果它们会导致在任一子节点中携带负权重的任何单个类，则也会忽略拆分。

4）**check\_input : *boolean, (default=True)***

允许绕过多个输入检查。除非您知道自己的操作，否则请勿使用此参数。

5）**X\_idx\_sorted : *array-like, shape = [n\_samples, n\_features], optional***

排序的训练输入样本的索引。如果在同一数据集上生成了许多树，则允许在树之间缓存排序。如果为None，则数据将在此处进行排序。除非您知道该怎么做，否则请勿使用此参数。

**返回：**

**self : *object***

1. [**get\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.get_params)([deep])

获取当前分类器的参数。

**参数：**

1）**deep : *boolean, optional***

如果是True，将返回此估算器的参数，并包含作为估算器的子对象。

**返回：**

**params : *mapping of string to any***

映射到其值的参数名称。

1. [**predict**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.predict)(X[, check\_input])

预测X的类或回归值。

对于分类模型，返回X中每个样本的预测类。对于回归模型，返回基于X的预测值。

**参数：**

1. **X : *array-like or sparse matrix of shape = [n\_samples, n\_features]***

输入样本。在内部，它将被转换为dtype=np.float32，并且如果向稀疏csr\_matrix提供稀疏矩阵。

1. **check\_input : *boolean, (default=True)***

允许绕过多个输入检查。除非您知道自己的操作，否则请勿使用此参数。

**返回：**

**y : *array of shape = [n\_samples] or [n\_samples, n\_outputs]***

预测的类或预测值。

1. [**predict\_log\_proba**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.predict_log_proba)(X)

预测输入样本X的类对数概率。

**参数：**

**X : *array-like or sparse matrix of shape = [n\_samples, n\_features]***

**返回：**

**p : *array of shape = [n\_samples, n\_classes], or a list of n\_outputs***

如果n\_outputs > 1，返回类似这样的数组。输入样本的类对数概率。类的顺序对应于属性classes\_中的顺序。

1. [**predict\_proba**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.predict_proba)(X[, check\_input])

预测输入样本X的类概率。

预测的类概率是叶中相同类的样本的分数。

**参数：**

**X : *array-like or sparse matrix of shape = [n\_samples, n\_features]***

**check\_input : *bool***

**返回：**

**p : *array of shape = [n\_samples, n\_classes], or a list of n\_outputs***

如果n\_outputs> 1，则为此类数组。输入样本的类概率。类的顺序对应于属性classes\_中的顺序。

1. [**score**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.score)(X, y[, sample\_weight])

返回给定测试数据和标签的平均准确度。

在多标签分类中，这是子集精度，这是一个严格的度量，因为您需要为每个样本正确预测每个标签集。

**参数：**

1）**X : *array-like, shape = (n\_samples, n\_features)***

测试样例。

2）**y : *array-like, shape = (n\_samples) or (n\_samples, n\_outputs)***

X的真实标签

3）**sample\_weight : *array-like, shape = [n\_samples], optional***

样本权重

**返回：**

**score : *float***

self.predict(X) wrt. y.的平均精度。

1. [**set\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.set_params)(\*\*params)

设置分类器的参数。

该方法适用于简单估计器以及嵌套对象（例如管道）。后者具有<component> \_\_ <parameter>形式的参数，因此可以更新嵌套对象的每个组件。

**返回：**

**Self**

## 5．2 项目实战

#加载必要的库

import pandas as pd

import random as rnd

import seaborn as sb

import matplotlib.pyplot as plt

%matplotlib inline#保证在每个cell执行后能够显示图片

#加载数据

train\_dataset = pd.read\_csv('train.csv')

test\_dataset = pd.read\_csv('test.csv')

#数据预处理

#定义数据预处理函数：删除无关的特征，以防止类似票价，票号等特征影响生存率的预测，必须在建模前去噪。

def data\_process(data):

#data["Fare"] = data["Fare"].fillna(data["Fare"].dropna().median())#以中值填补缺失值

data["Age"] = data["Age"].fillna(data["Age"].dropna().median())

data = data.drop(['Fare', 'Ticket', 'Cabin', 'Embarked', 'Name'], axis=1)#丢弃无关列信息

data.loc[data["Sex"] == "male", "Sex"] = 0#男性标识为0，女性标识为1

data.loc[data["Sex"] == "female", "Sex"] = 1

data.loc[data['Age'] <= 16, 'Age'] = 0#连续值处理

data.loc[(data['Age'] > 16) & (data['Age'] <= 32), 'Age'] = 1

data.loc[(data['Age'] > 32) & (data['Age'] <= 48), 'Age'] = 2

data.loc[(data['Age'] > 48) & (data['Age'] <= 64), 'Age'] = 3

data.loc[data['Age'] > 64, 'Age'] = 4

return data

train\_dataset = data\_process(train\_dataset)

train\_dataset.head()

#决策树算法建模

from sklearn import tree

target = train\_dataset["Survived"].values

features = train\_dataset[["Pclass", "Sex", "Age", "SibSp", "Parch"]].values

decision\_tree = tree.DecisionTreeClassifier(random\_state = 1)

decision\_tree\_ = decision\_tree.fit(features, target)

decision\_tree\_.score(features, target)