决策树算法

决策树是一种基本的分类方法，当然也可以用于回归。我们一般只讨论用于分类的决策树。决策树模型呈树形结构。在分类问题中，表示基于特征对实例进行分类的过程，它可以认为是if-then规则的集合。在决策树的结构中，每一个实例都被一条路径或者一条规则所覆盖。通常决策树学习包括三个步骤：**特征选择**、决策树的生成和决策树的修剪

> 优点：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，**可以处理逻辑回归等不能解决的非线性特征数据**
>
> 缺点：可能产生过度匹配问题适
>
> 用数据类型：数值型和标称型

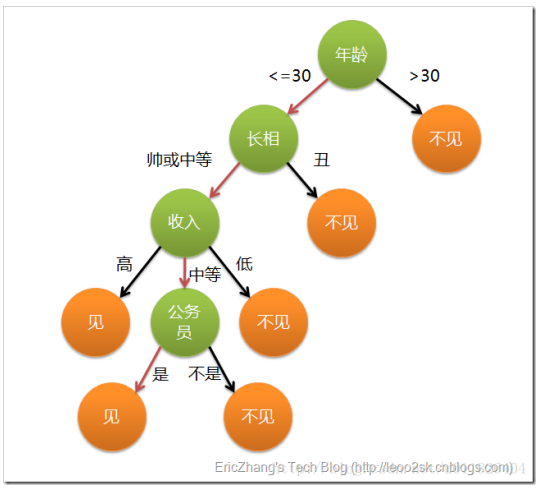
通过信息增益生成的决策树结构，更加明显、快速的划分类别。下面介绍scikit-learn中API的使用

1 决策树基本概念

顾名思义，决策树是基于树结构来进行决策的，在网上看到一个例子十分有趣，放在这里正好合适。现想象一位捉急的母亲想要给自己的女娃介绍一个男朋友，于是有了下面的对话：

> 女儿：多大年纪了？
>
> 母亲：26。
>
> 女儿：长的帅不帅？
>
> 母亲：挺帅的。
>
> 女儿：收入高不？
>
> 母亲：不算很高，中等情况。
>
> 女儿：是公务员不？
>
> 母亲：是，在税务局上班呢。
>
> 女儿：那好，我去见见。

这个女孩的挑剔过程就是一个典型的决策树，即相当于通过年龄、长相、收入和是否公务员将男童鞋分为两个类别：见和不见。假设这个女孩对男人的要求是：30岁以下、长相中等以上并且是高收入者或中等以上收入的公务员，那么使用下图就能很好地表示女孩的决策逻辑（即一颗决策树）。



在上图的决策树中，决策过程的每一次判定都是对某一属性的“测试”，决策最终结论则对应最终的判定结果。一般一颗决策树包含：一个根节点、若干个内部节点和若干个叶子节点，易知：

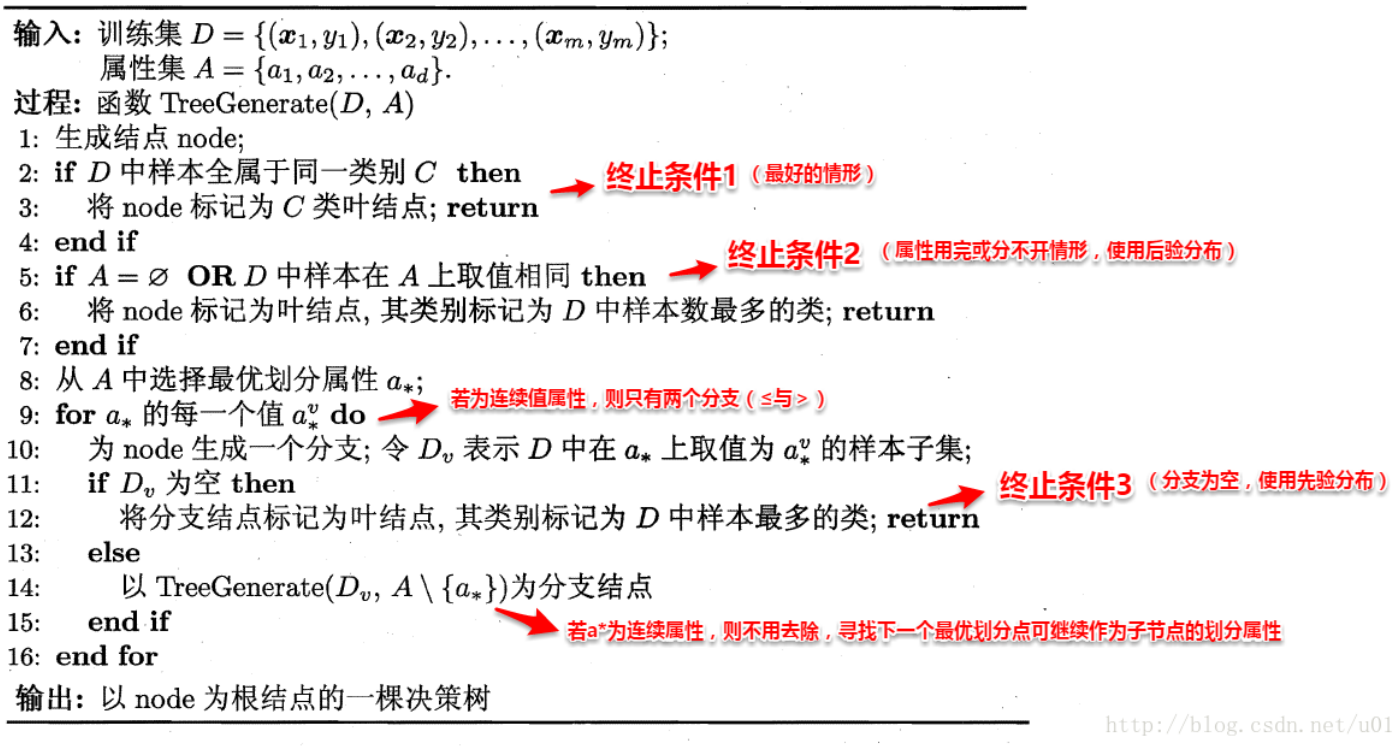
* 每个非叶节点表示一个特征属性测试。
* 每个分支代表这个特征属性在某个值域上的输出。

* 每个叶子节点存放一个类别。

* 每个节点包含的样本集合通过属性测试被划分到子节点中，根节点包含样本全集。

2 决策树的构造

决策树的构造是一个递归的过程，有三种情形会导致递归返回：(1) 当前结点包含的样本全属于同一类别，这时直接将该节点标记为叶节点，并设为相应的类别；(2) 当前属性集为空，或是所有样本在所有属性上取值相同，无法划分，这时将该节点标记为叶节点，并将其类别设为该节点所含样本最多的类别；(3) 当前结点包含的样本集合为空，不能划分，这时也将该节点标记为叶节点，并将其类别设为父节点中所含样本最多的类别。算法的基本流程如下图所示：



可以看出：决策树学习的关键在于如何选择划分属性，不同的划分属性得出不同的分支结构，从而影响整颗决策树的性能。属性划分的目标是让各个划分出来的子节点尽可能地“纯”，即属于同一类别。因此下面便是介绍量化纯度的具体方法，决策树最常用的算法有三种：ID3，C4.5和CART。

2.1 ID3算法

ID3算法使用信息增益为准则来选择划分属性，“信息熵”(information entropy)是度量样本结合纯度的常用指标，假定当前样本集合D中第k类样本所占比例为pk，则样本集合D的**信息熵**定义为：

$$\mathrm{Ent}(D)=-\sum_{k=1}^{|\mathcal{Y}|}p_k\log_2 p_k\ .$$

值越大表示越混乱，易知只有一个类别时，信息熵为0

假定通过属性划分样本集D，产生了V个分支节点，v表示其中第v个分支节点，易知：分支节点包含的样本数越多，表示该分支节点的影响力越大。故可以计算出划分后相比原始数据集D获得的“**信息增益**”（information gain）。

$$\mathrm{Gain}(D,a)=\mathrm{Ent}(D)-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v)\ .$$

信息增益越大，表示使用该属性划分样本集D的效果越好，因此ID3算法在递归过程中，每次选择最大信息增益的属性作为当前的划分属性。

2.2 C4.5算法

ID3算法存在一个问题，就是偏向于取值数目较多的属性，例如：如果存在一个唯一标识，这样样本集D将会被划分为|D|个分支，每个分支只有一个样本，这样划分后的信息熵为零，十分纯净，但是对分类毫无用处。因此C4.5算法使用了“增益率”（gain ratio）来选择划分属性，来避免这个问题带来的困扰。首先使用ID3算法计算出信息增益高于平均水平的候选属性，接着C4.5计算这些候选属性的增益率，**增益率**定义为：

$$\mathrm{Gain\_ratio}(D,a)=\frac{\mathrm{Gain}(D,a)}{\mathrm{IV}(a)}\ ,$$

其中

$$\mathrm{IV}(a)=-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\log_2\frac{|D^v|}{|D|}$$

当α属性的取值越多时
IV(α)值越大

2.3 CART算法

CART决策树使用“基尼指数”（Gini index）来选择划分属性，基尼指数反映的是从样本集D中随机抽取两个样本。

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**

，因此Gini(D)越小越好，基尼指数定义如下：

$$\begin{aligned}\mathrm{Gini}(D)&=\sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k}p_k p_{k'}\\&=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2\ .\end{aligned}$$

任取两个样本类标不一致的概率

越小表示集合越纯

进而，使用属性 α 划分后的基尼指数为：

$$\text{Gini\_index}(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\text{Gini}(D^v)\ .$$

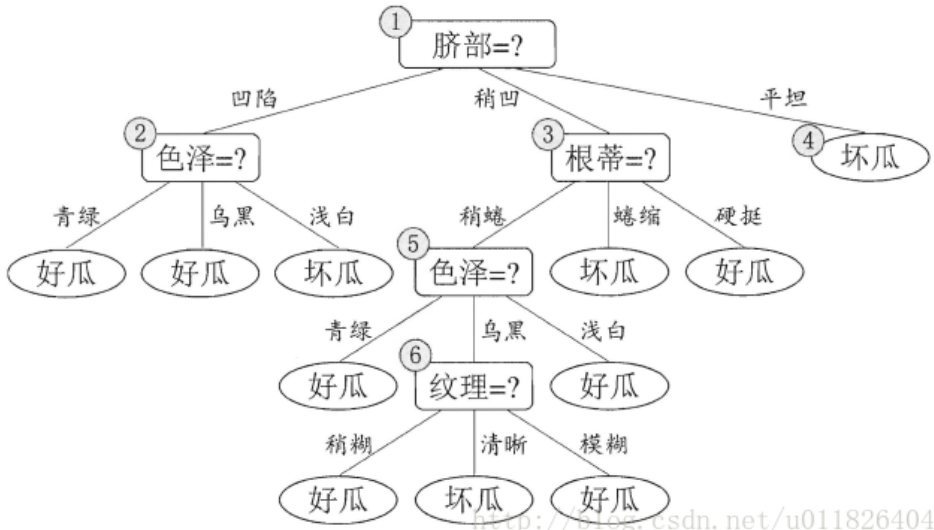
故选择基尼指数最小的划分属性

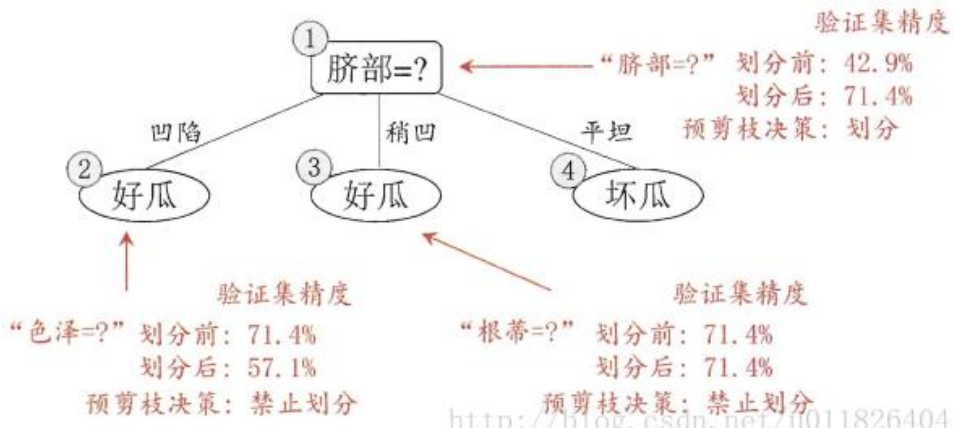
3 剪枝处理

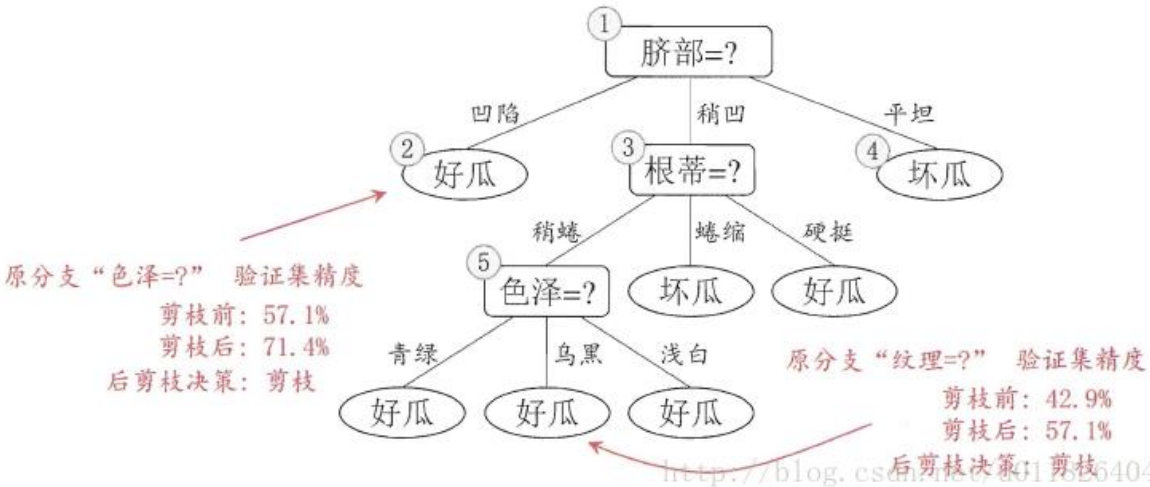
从决策树的构造流程中我们可以直观地看出：不管怎么样的训练集，决策树总是能很好地将各个类别分离开来，这时就会遇到之前提到过的问题：过拟合（overfitting），即太依赖于训练样本。剪枝（pruning）则是决策树算法对付过拟合的主要手段，剪枝的策略有两种如下：

* 预剪枝（prepruning）：在构造的过程中先评估，再考虑是否分支。
* 后剪枝（post-pruning）：在构造好一颗完整的决策树后，自底向上，评估分支的必要性。

评估指的是性能度量，即决策树的泛化性能。之前提到：可以使用测试集作为学习器泛化性能的近似，因此可以将数据集划分为训练集和测试集。预剪枝表示在构造数的过程中，对一个节点考虑是否分支时，首先计算决策树不分支时在测试集上的性能，再计算分支之后的性能，若分支对性能没有提升，则选择不分支（即剪枝）。后剪枝则表示在构造好一颗完整的决策树后，从最下面的节点开始，考虑该节点分支对模型的性能是否有提升，若无则剪枝，即将该节点标记为叶子节点，类别标记为其包含样本最多的类别。







上图分别表示不剪枝处理的决策树、预剪枝决策树和后剪枝决策树。预剪枝处理使得决策树的很多分支被剪掉，因此大大降低了训练时间开销，同时降低了过拟合的风险，但另一方面由于剪枝同时剪掉了当前节点后续子节点的分支，因此预剪枝“贪心”的本质阻止了分支的展开，在一定程度上带来了欠拟合的风险。而后剪枝则通常保留了更多的分支，因此采用后剪枝策略的决策树性能往往优于预剪枝，但其自底向上遍历了所有节点，并计算性能，训练时间开销相比预剪枝大大提升。

4 决策树优缺点分析

决策树的一些优点是：

- 简单的理解和解释。树木可视化。
- 需要很少的数据准备。其他技术通常需要数据归一化，需要创建虚拟变量，并删除空值。但请注意，此模块不支持缺少值。
- 使用树的成本（即，预测数据）在用于训练树的数据点的数量上是对数的。

决策树的缺点包括：

- 决策树学习者可以创建不能很好地推广数据的过于复杂的树。这被称为过拟合。修剪（目前不支持）的机制，设置叶节点所需的最小采样数或设置树的最大深度是避免此问题的必要条件。
- 决策树可能不稳定，因为数据的小变化可能会导致完全不同的树被生成。通过使用合作的决策树来减轻这个问题。