# 第四章　决策树

## 0.1　基本流程

一棵决策树（decision treee）可以分成三个部分：叶节点，非叶节点，分支。叶节点对应决策结果，也即分类任务中的类别标记；非叶节点（包括根节点）对应一个判定问题（某属性 =?）；分支对应父节点判定问题的不同答案（可能的属性值），可能连向一个非叶节点的子节点，也可能连向叶节点。

决策树生成是一个**递归过程**：

1. 传入训练集和属性集
2. 生成一个新节点
3. 若此时数据集中所有样本都属于同一类，则把新节点设置为该类的叶节点，然后**返回** [1]。
4. 若此时属性集为空，或者数据集中所有样本在属性集余下的所有属性上取值都相同，无法进一步划分，则把新节点设置为叶节点，类标记为数据集中样本数最多的类，然后**返回** [2]
5. 从属性集中选择一个最优划分属性
   - 为该属性的每个属性值生成一个分支，并按属性值划分出子数据集
   - 若分支对应的子数据集为空，无法进一步划分，则直接把子节点设置为叶节点，类标记为父节点数据集中样本数最多的类，然后**返回** [3]
   - 将子数据集和去掉了划分属性的子属性集作为算法的传入参数，继续生成该分支的子决策树。

3 处返回中的第 2 处和第 3 处设置叶节点的类标记原理有所不同。第 2 处将类标记设置为当前节点对应为数据集中样本数最多的类，这是利用当前节点的**后验分布**；第 3 处将类标记设置为为父节点数据集中样本数最多的类，这是把父节点的样本分布作为当前节点的**先验分布**。

先验概率（prior probability）：在获得某些信息之前对某个变量 p 的概率分布情况进行猜测。先验概率仅仅依赖主观上的经验估计。后验概率（posterior probability）：在相关证据或背景给定并纳入考虑之后的条件分布，即关于参数 $\theta$ 在给定的证据信息 $X$ 下的后验概率是：$P(\theta|X)$。似然函数（likelihood function）：对于结果 $X$，在参数集合 $\theta$ 上的似然，就是在参数给定条件下观察到 $X$ 值的条件分布：$P(X|\theta)$。

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

先验就是设定一种情形，似然就是看这种情形下发生的可能性，两者合起来就是后验的概率。

$$Posterior\ Probability \propto Likelihood\ \times Prior\ Probability$$

后验分布正比于先验分布 × 似然函数。$\propto$ 表示正比于。

## 0.2　划分选择

划分的目标是：尽可能让每个划分子节点中都是同一类别，即结点的纯度越高越好。

### 0.2.1 信息增益– ID3

**信息熵**（information entropy）：

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

其中 $|\mathcal{Y}|$ 为类别集合，$p_k$ 为该类样本占样本总数的比例。信息熵越大，表示样本集的混乱程度越高，纯度越低。

信息增益（information gain）是 ID3 算法采用的选择准则，定义如下：

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

它描述的是按某种属性划分后纯度的提升，**信息增益越大，代表用属性 $a$ 进行划分所获得的纯度提升越大**。其中 $V$ 表示属性 $a$ 的属性值集合，$D^v$ 表示属性值为 $v$ 的数据子集。求和项也称为条件熵，可以理解为它是先求出每个数据子集的信息熵，然后按每个数据子集占原数据集的比例来赋予权重，比例越大，对提升纯度的帮助就越大。

### 0.2.2 增益率– C4.5

增益率（gain ratio）是 C4.5 算法采用的选择准则，定义如下：

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

其中，

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

IV 称为属性的固有值（intrinsic value），它的定义和信息熵是类似的，信息熵衡量的是样本集在类别上的混乱程度，而固有值衡量的是样本集在某个属性上的混乱程度。**固有值越大，则该属性混乱程度越高，可能的取值越多。**

之所以要定义增益率是为了**避免模型过份偏好用取值多的属性作划分**。这是使用信息增益作准则非常容易陷入的误区，比方说每个样本都有一个“编号”属性，这个属性的条件熵肯定是最小的，但如果选择了该属性作为根节点，那么构建出的决策树就没有任何意义了，因为这个模型根本不具备泛化性能。

C4.5 并非直接选择增益率最高的属性，它使用了一个**启发式**：先从属性集中找到信息增益高于平均水平的属性作为候选，然后再比较这些候选属性的增益率，从中选择增益率最高的。

### 0.2.3 基尼指数–CART

基尼指数（Gini index）是 CART 算法采用的选择准则，定义如下：基尼值：

$$Gini(D) = \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2$$

其中 $p_k$ 为该类样本占样本总数的比例。

基尼指数：

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

基尼值是另一种衡量样本集纯度的指标。反映的是**从一个数据集中随机抽取两个样本，其类别标志不同的概率**。基尼值越小，样本集的纯度越高。

## 0.3 剪枝处理

剪枝（pruning）是决策树学习算法应对**过拟合**的主要手段。判断剪枝是否有用即判断剪枝后模型的泛化性能有没有提升。

### 0.3.1 预剪枝

预剪枝（prepruning）是在决策树生成的过程中，对每个节点在划分前先进行估计，若当前节点的划分不能带来决策树泛化性能提升（划分后在测试集上错得更多了 / 划分前后在测试集上效果相同），就停止划分并将当前节点标记为叶节点。

### 0.3.2 后剪枝

后剪枝（postpruning）是先从训练集生成一颗完整的决策树，然后自底向上地逐个考察非叶节点，若将该节点对应**的子树替换为叶节点**能带来决策树泛化性能的提升，则将该子树替换为叶节点。实际任务中，即使没有提升，只要不是性能下降，一般也会剪枝，因为根据奥卡姆剃刀准则，简单的模型更好。

特别地，只有一层划分（即只有根节点一个非叶节点）的决策树称为决策树桩（decision stump）。

### 0.3.3 优缺点

预剪枝是一种贪心策略，因为它在决策树生成时就杜绝了很多分支展开的机会，所以不但降低了过拟合的风险，同时也 ** 显著减少了模型的训练时间开销和测试时间开销 **。但是这种贪心策略有**可能导致欠拟合**，因为有可能当前划分不能提升模型的泛化性能，但其展开的后续划分却会显著提升泛化性能，在预剪枝中这种可能被杜绝了。

后剪枝是种比较保守的策略，欠拟合的风险很小，泛化性能往往优于预剪枝的决策树。但是由于后剪枝是在生成了完整决策树后，自底向上对所有非叶节点进行考察，所以**训练时间开销**要比未剪枝决策树和预剪枝决策树都大得多。

## 0.4 连续与缺省值

### 0.4.1 连续值处理

前面线性模型已经谈到了离散属性连续化，而决策树模型需要的则是**连续属性离散化**，因为决策树每次判定只能做有限次划分。最简单的一种离散化策略是 C4.5 算法采用的二分法（bi-partition）。

给定一个包含连续属性 $a$ 的数据集，并且 $a$ 在数据集中有 $n$ 个不同取值，我们先把属性 $a$ 的 $n$ 个属性值从小到大进行排序。所谓“二分”是指将这些属性值分为两个类别（比如把身高这一属性分为高于 170 和低于 170 两个类别）。

这就产生了一个新问题，怎么找到合适的划分点（例如上面例子的 170）呢?

在对连续属性值排序完之后，由于有 $n$ 个不同取值，取**每两个取值的平均值作为划分点**的话，就有 $n-1$ 个候选划分点。按照相应准则（比方说用 ID3 算法的话就是信息增益）进行 $n-1$ 次判断。每次拿出一个候选划分点，把连续属性分为两类，转换为离散属性，然后基于这个基础计算准则，最终选出一个最优的属性值划分点。

注意: 和离散属性不同，连续属性用于当前节点的划分后，其**后代节点依然可以使用该连续属性进一步划分**。例如，当前节点用身高低于 170 划分了，那么它的后代节点还可以用身高低于 160 来进一步划分。

### 0.4.2 缺失值处理

假设数据集为 $D$，有缺失值的属性为 $a$，令 $\tilde{D}$ 表示 $D$ 中没有缺失属性 $a$ 的样本子集。定义变量：

$$\rho = \frac{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in D} w_{\mathbf{x}}}$$
$$\tilde{p_k} = \frac{\sum_{\mathbf{x}\in\tilde{D}_k} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}, \quad (1 \le k \le |\mathcal{Y}|)$$
$$\tilde{r_v} = \frac{\sum_{\mathbf{x}\in\tilde{D}^v} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}, \quad (1 \le v \le V)$$

**$\rho$ 表示无缺失值样本所占的比例；$\tilde{p_k}$ 表示无缺失值样本中第 $k$ 类所占的比例；$\tilde{r_v}$ 表示无缺失值样本中在属性 $a$ 上取值 $a^v$ 的样本所占的比例。**

注意：这里的 $w_{\mathbf{x}}$ 表示样本的权值，它是含缺失值样本参与建模的一种方式。在根节点处初始时，所有样本 $\mathbf{x}$ 的权重都为 1。

接下来重新定义信息熵和信息增益，推广到样本含缺失值的情况：

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

$$Gain(D,a) = \rho \times Gain(\tilde{D},a) = \rho \times (Ent(\tilde{D}) - \sum_{v=1}^{V} \tilde{r_v} Ent(\tilde{D}^v))$$

按照新的定义来计算包含缺失值的属性的信息增益，然后和其他属性的信息增益相比，选出最优的划分属性。

假设有一个包含缺失值的属性被计算出是最优划分属性，那么我们就要按该属性的不同取值划分数据集了。缺失该属性值的样本怎么划分呢？答案是**按概率划分**，这样的样本会被**同时**划入所有子节点，并且其权重更新为对应的 $\tilde{r_v} \dot{w}_{\mathbf{x}}$。

可以把无缺失值的决策树建模想象为各样本权值恒为 1 的情形，它们只对自己所属的属性值子集作贡献。而样本含缺失值时，它会以不同的概率对所有属性值子集作贡献。

## 0.5 多变量决策树

前面提到的决策树都是单变量决策树（univariate decision tree），即在每个节点处做判定时都只用到一个属性。它有一个特点，就是形成的分类边界都是轴平行（axis-parallel）的，即分类边界由若干个与坐标轴平行的分段组成。

如果把属性都当作坐标空间中的坐标轴，由于我们建模时假设样本的各属性之间是没有关联的，所以**各坐标轴是相互垂直的**。而决策数每次只取一个确定的属性值来划分，就等同于画一个垂直于该属性坐标轴的超平面（只有两个属性时就是一条线），它与其他坐标轴都是平行的，这就是轴平行，最终由多个与坐标轴平行的超平面组成分类边界。

这样有一个弊端就是，如果真实分类边界特别复杂，就需要画出很多超平面（线），在预测时就需要继续大量的属性测试（遍历决策树）才能得到结果，预测时间开销很大。

多变量决策树（multivariate decision tree），不再是选择单个最优划分属性作为节点，而是试图寻找一个**最优的多属性的线性组合**作为节点，它的每个非叶节点都是一个形如 $\sum_{i=1}^{d} w_i a_i = t$ 的线性分类器。多变量决策树的决策边界能够斜着走，甚至绕曲线走，从而用更少的分支更好地逼近复杂的真实边界。