第五章　决策树

5.1　简介

5.1 简 介

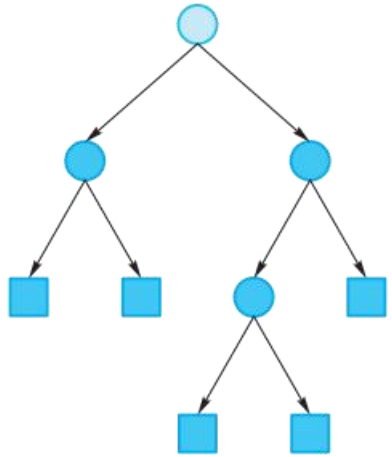
- 相较于前两种方法, 决策树通过树状结构进行决策, 每个节点表示对一个特征的测试, 每个分支代表一个测试结果, 每个叶节点存储一个输出值. 决策树算法可以处理分类特征和数值特征, 不需要对数据进行特殊的缩放, 适用于需要可解释性的情况.
- 决策树方法最早产生于 20 世纪 60 年代, 其中 CART 算法是决策树最经典和最主要的算法. CART 算法 (classification and regression tree) 是 Breiman等[67] 在 1984 年提出来的一种非参数方法, 它可以用于解决分类问题 (预测定性变量, 或者说当因变量是离散变量时), 又可以用于回归问题 (预测定量变量, 或者说当因变量是连续变量时), 分别称为分类树 (classification tree) 和回归树 (regression tree). CART 算法的基本思想是一种二分递归分割方法,在计算过程中充分利用二叉树, 在一定的分割规则下将当前样本集分割为两个子样本集, 使得生成的决策树的每个非叶子节点都有两个分裂, 这个过程又在子样本集上重复进行, 直至无法再分成叶子节点为止. 在本章中, 我们介绍的内容主要包括：决策树的基本概念、回归树和分类树的建模过程、防止决策树过拟合的方法及实施决策树相关的 Python 和 R 语言程序.

5.2　决策树的基本原理

5.2 决策树的基本原理

- 决策树 (decision tree) 是一种常见的机器学习方法, 决策树模型呈树形结构, 采用自顶向下递归的方法. 决策树包含三种节点, 分别是根节点、内部节点和叶子节点. 刚开始构建模型时, 全部样本组成的节点, 没有入边, 只有出边, 称为根节点 (root node); 不再继续分裂的节点称为树的叶子节点 (leafnode), 没有出边, 只有入边, 叶子节点的个数决定了决策树的规模和复杂程度; 根节点和叶子节点之外的节点都称作内部节点 (internal node), 内部节点既有入边, 又有出边.

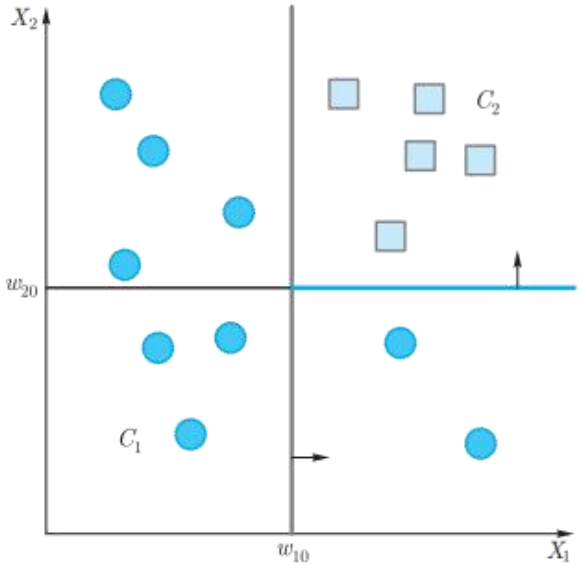
- 决策树模型结构如图 5.1 所示, 一棵决策树一般包含一个根节点 (最上方圆形图标)、若干个内部节点 (第二层及以下圆形图标) 和若干个叶子节点 (方形图标). 叶子节点对应于决策结果, 每个内部节点对应一个自变量, 每个内部节点包含的样本集合根据自变量测试的结果被划分到它的子节点中, 根节点包含全部训练样本. 根节点在一定的分割规则下被分割成两个子节点, 这个过程在子节点上重复进行, 直至无法再分为叶子节点为止. 从根节点到每个叶子节点的路径对应了一个判定测试序列.

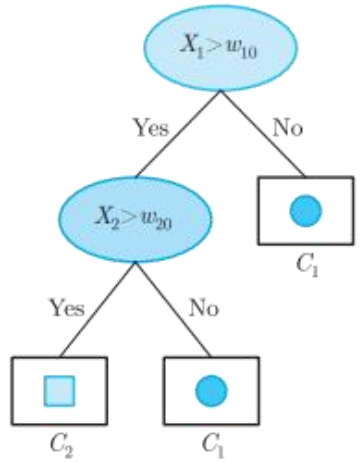


5.2 决策树的基本原理

- 决策树算法遵循自顶向下、分而治之的策略, **具体步骤为**
  - (1) 选择最好的自变量作为测试自变量并创建树的根节点;
  - (2) 为测试自变量每个可能的取值产生一个分支;
  - (3) 将训练样本划分到适当的分支形成子节点;
  - (4) 对每个子节点, 重复上面的过程, 直到所有的节点都是叶子节点.

- 若考虑自变量只有两维的情况, 将决策树分类过程在二维的坐标轴中画出, 如图 5.2 所示. 则可以发现, 决策树实际上就是对自变量空间的划分, 且划分区域的数目就是叶子节点的数目, 如图 5.2 中的粗线.





5.2 决策树的基本原理

- 决策树的建立过程可以概括为以下两个关键环节：
  - (1) 将自变量空间 (即 $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)^{\mathrm{T}}$ 的所有可能取值构成的集合) 分割成 T 个互不重叠的叶子节点$R_1, \cdots, R_T$；
  - (2) 对落入节点 R_t 的每个观测点, 其预测值为 R_t 节点中训练集的因变量值的众数 (分类树), 或者平均数 (回归树).
- 决策树生长是一个递归的过程, 因此在算法中需要包含跳出递归的条件.直观地看, 决策树停止生长的条件有以下三种情况, 满足以下三个条件之一就能使决策树停止生长：
  - (1) 当前节点包含的样本全部属于同一类别;
  - (2) 当前自变量集为空, 或所有样本在所有自变量上取值相同;
  - (3) 当前节点包含的样本集合为空.
- 决策树算法有以下几条**优点**：
  - (1) 计算量相对小, 训练速度快;
  - (2) 易理解、解释性强;
  - (3) 不需要任何先验假设;

5.2 决策树的基本原理

- (4) 可以处理连续变量和离散变量 (如性别);
- (5) 可以处理缺失值和具有尺度不变性, 即对自变量做一个单调变换, 不改变树的生成结果.

■ 决策树算法包括如下**缺点**:

- (1) 决策树算法可以创建很复杂的树, 但容易导致过拟合, 可以通过剪枝等手段缓解;
- (2) 决策树算法训练结果方差大、不稳定, 数据很小的扰动可能得到完全不同的分裂结果, 有可能是完全不同的决策树, 这个缺点可以通过集成学习来解决

■ 过拟合 (overfitting) 是指机器学习模型在训练数据上表现得过于优越,以至于在未见过的新数据上表现不佳的现象. 简而言之, 过拟合发生时, 模型过度适应了训练数据的噪声和细节, 而没有正确地捕捉到数据中的真实模式和普遍规律.

■ 具体而言, 过拟合通常表现为以下一些特征:

- (1) 训练数据拟合良好: 模型在训练数据上表现良好, 准确度高, 损失函数低;
- (2) 测试数据表现差: 在未见过的测试数据上, 模型的性能较差, 预测结果不够准确;

5.2 决策树的基本原理

- (3) 模型过于复杂：模型可能过度灵活, 以至于能够适应训练数据中的每一个细节, 包括噪声和异常值;
- (4) 泛化能力下降：模型在新数据上的泛化能力减弱, 不能很好地适应不同的数据分布.

■ 防止过拟合也有一些常用的方法, 比如：

- (1) 训练数据增强：增加训练数据量, 有助于提高模型的泛化能力;
- (2) 交叉验证：使用交叉验证来评估模型在不同数据集上的性能, 有助于发现过拟合问题;
- (3) 正则化：引入正则化项, 如 L_1 正则化或 L_2 正则化, 以限制模型参数的大小;
- (4) 特征选择：选择最重要的特征, 避免使用过多不相关或冗余的特征;
- (5) 模型简化：选择较简单的模型结构, 避免使用过于复杂的模型;
- (6) 早停：在训练过程中监测模型性能, 在验证数据上性能不再提升时停止训练, 防止过拟合.

■ 过拟合是机器学习中需要注意的重要问题, 合适的防范手段能够提高模型的泛化性能, 使其更好地适应新的未见数据.

5.3　分类树与回归树

5.3 分类树与回归树

- 对于决策树而言, 根据预测变量的类型不同, 可分为不同的决策树. 若目标变量是离散的, 则为分类树, 若目标变量是连续的, 则为回归树.

5.3.1 分类树

- 我们首先介绍分类树. 当因变量为定性变量即离散型变量时, 可建立分类树模型. 分类树是一种特殊的分类模型, 是一种直接以树的形式表征的非循环图. 它的建模过程就是自动选择分裂变量, 以及根据这个变量进行分裂的条件.
- 假设数据 $\boldsymbol{X}_i = (X_{i1}, \cdots, X_{ip})^{\mathrm{T}}$ 包含 p 个输入变量和一个离散型的因变量 $\boldsymbol{Y} \in (1, 2, \cdots, K)$, 样本量为 n. 现在想把数据分成 T 个区域 (或称为节点), 第 t 个节点 R_t 的样本量为 $n_t (t = 1, 2, \cdots, \mathrm{T})$.
- 建立分类树的过程可以用如下**算法**表示:

  ▶ **1. 分支**

  采用递归二叉分裂法在训练集中生成一棵分类树. 递归二叉分裂法是指是从树顶端开始依次分裂自变量空间, 每个分裂点都产生两个新的分裂, 并且每次分裂选取最优分裂方案. 最优的分裂方案是使得 T 个节点的不纯度减少到最小, 衡量节点不纯度的指标在 5.4 节详细介绍.

  ▶ **2. 剪枝**

  对生成的回归树进行剪枝, 得到一系列最优子树, 剪枝在 5.5 节详细介绍.

5.3.1 分类树

3. 预测

令 $\hat{P}_{tk}=\frac{1}{n}\sum_{i\in R_t} I(Y_i=k)$ 表示在叶子节点 t 中第 k 类样本点的比例, 则预测叶子节点 t 的类别为

$$\hat{t}_m=\text{argmax}_k \hat{P}_{tk}.$$

即叶子节点 t 中类别最多的一类.

5.3.2 回归树

- 当因变量为连续型变量时, 可建立回归树模型. 回归树和分类树的思想类似, 首先是在分支准则上有差异, 即衡量节点不纯度的指标不一样, 其次是预测准则上存在差异, 对于分类树, 将落在该叶子节点的观测点的最大比例类别作为该叶子节点预测值, 而对于回归树, 则是将落在该叶子节点的观测点的平均值作为该叶子节点预测值.
- 回归树的数据结构与分类树类似, 唯一区别是回归树解决的是连续型的因变量 Y , 并且回归树算法过程跟分类树类似.

  ▶ **1. 分支**

  采用递归二叉分裂法在训练集中生成一棵回归树. 最优的分裂方案是使得 T 个节点的不纯度减少到最小, 其中, 衡量节点不纯度的指标为样本均方误差.

  ▶ **2. 剪枝**

  对生成的分类树进行剪枝, 得到一系列最优子树, 剪枝在 5.5 节详细介绍.

5.3.2 回归树

3. 预测

得到节点后, 可以确定某一给定预测数据所属的节点, 回归树用这一节点的训练集的因变量的平均值作为预测值:

$$\hat{Y}_t = \frac{1}{n_t} \sum_{i \in R_t} Y_i .$$

5.4　分支条件

5.4 分支条件

- 现在再来讨论该如何构建节点, 即对决策树进行分支. 理论上, 我们可以将节点所对应的区域形状作任意分割, 但出于模型的简化和可解释性考虑,一般只将区域划分为高维矩形. 不过, 若要将自变量空间划分为矩形区域的所有可能性都进行考虑, 这在计算上是不可行的. 所以, 我们对区域的划分一般采用一种自上而下(top-down)、贪婪 (greedy) 的方法：递归二叉分裂(recursive binary splitting). 自上而下指的是从树顶端开始依次分裂自变量空间, 每个分裂点都产生两个新的分裂. 贪婪指在建立树的每一步中, 最优分裂确定仅限于某一步进程, 而不是针对全局, 选择那些能够在未来进程中构建出更好的树的分裂点. 对每一个节点重复以上过程, 寻找继续分割数据集的最优自变量和最优分裂点. 此时被分割的不再是整个自变量空间, 而是之前确定的两个区域之一. 这一过程不断持续, 直到符合某个分裂准则再停止,譬如, 当叶子节点包含的观测值个数低于某个最小值时, 分裂停止.

- 如何确定最优的分裂方案？我们基于不纯度的减少来作为分裂准则, 即通过最小化节点不纯度来确定最优分裂变量和最优分裂点. 对于分类树和回归树有不同的衡量节点不纯度的指标, 我们将在后面的分类树和回归树部分进行具体的描述.

5.4 分支条件

- 决策树算法的关键是划分自变量的选择, 决策树自变量划分方法很多, 比较经典的有适用于分类树的信息增益、增益率和基尼指数, 以及适用于回归树的均方误差.

5.4.1 信息熵

- 在介绍信息增益之前, 我们先引入信息量和信息熵的概念.
- 假设随机变量 X 的概率分布为 $P(x)$, 则任意一事件 $X=x$ 的信息量可以表示为 $h(x)=-\log_2 P(X=x)$, 其中 $P(X=x)$ 表示该事件发生的概率. 某事件发生的概率越小, 该事件的信息量越大.
- 信息熵是度量样本集合纯度最常用的一种指标, 也是一种不确定性的度量. 1948 年, 香农 (Shannon)[68] 在他著名的《通信的数学原理》论文中指出: "信息是用来消除随机不确定性的东西", 并提出了"信息熵"的概念 (借用了热力学中熵的概念), 来解决信息的度量问题.
- 对于随机变量 X 而言, 信息熵可表示为

$$H(X)=-\sum_{X_i\in\mathcal{X}} P(X=X_i)\ln P(X=X_i),$$

  - 其中 $\mathcal{X}$ 表示所有可测事件的集合.

5.4.1 信息熵

- 特别地, 若假设两点分布中某一点发生的概率是 θ, 则其信息熵为

$$H(\boldsymbol{X})=-\sum_{\boldsymbol{X}_i\in\mathcal{X}}P(\boldsymbol{X}=\boldsymbol{X}_i)\ln P(\boldsymbol{X}=\boldsymbol{X}_i)=-\theta\ln\theta-(1-\theta)\ln(1-\theta).$$

- 若信息熵退化为定值则熵最小为 0; 若随机变量是连续型随机变量且取值在有限区间内 (即随机变量密度函数在有限区间内大于零), 则均匀分布的熵最大; 随机变量是连续型随机变量且取值在无限区间内, 通过拉格朗日乘子法可以推导出正态分布的熵最大.

- 下面考虑响应变量是离散型随机变量的情况下的信息熵. 即 $Y\in\{1,2,\cdots,K\}$ 的 K 分类问题.
- 对于离散型响应变量集合 $\boldsymbol{Y}=(Y_1,\cdots,Y_n)^{\mathrm{T}}$ 来说, 当前集合中第 k 类样本所占的比例为, $\hat{P}_k=\frac{1}{n}\sum_{i=1}^{n}I(Y_i=k)$, 则信息熵的计算公式为

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\hat{P}_k\log_2\hat{P}_k. \tag{5.4.1}$$

5.4.1 信息熵

- 信息熵越大, 意味着不确定性越大; 信息熵越小, 则不确定性越小. 例如,对于一盏灯有两种可能的状态：开和关, 假设样本集合中一半是开的样本, 一半是关的样本, 那么信息熵为

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\hat{P}_k\log_2\hat{P}_k=-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1.$$

  - 在此情况下, 信息熵很大, 说明灯的两种状态不确定性很大.

- 假如这盏灯出现故障, 无法正常打开, 那么此时信息熵为

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\hat{P}_k\log_2\hat{P}_k=-\left(\frac{1}{1}\log_2\frac{1}{1}\right)=0.$$

  - 在此情况下, 灯的状态是确定的, 因此信息熵很小.

5.4.2 信息增益

- 有了信息熵的概念, 我们引入信息增益, 信息增益是针对某一种自变量而言的, 假设为 X(一维).
- 假设使用自变量 X 的样本集$\boldsymbol{X}=(X_1,\cdots,X_n)^{\mathrm{T}}$ 对分类树划分出 M 个区域 $R_1,\cdots,R_M$(在二叉分裂时, $M=2$), 其中叶子节点 R_m 中含有样本数为n_m, 简记为 $|R_m|=n_m$. 假设节点 R_m 中类 k 的观测比例为

$$\hat{P}_{mk}=\frac{1}{n_m}\sum_{i\in R_m}I\left(Y_i=k\right).$$

- 假设自变量是离散变量, M 个节点是由自变量 X 的 M 个可能的取值划分出来的. 若自变量是连续变量, 我们可以从连续变量取值范围上取几个点对样本进行划分, 此时被划分的样本集可等价理解成离散变量样本.
- 接下来, 计算出用离散变量样本集$\boldsymbol{X}$对样本集$\boldsymbol{Y}$ 进行再次划分所获得的信息增益, 假设$\boldsymbol{X}$取自集合 $\{1,2,\cdots,M\}$, $\boldsymbol{Y}$ 取自集合 $\{1,2,\cdots,K\}$.

5.4.2 信息增益

- 信息增益 (information gain) 表示自变量 X 使目标不确定性减少的程度.计算公式如下

$$\Delta = H(\boldsymbol{Y}) - \sum_{m=1}^{M} \frac{n_m}{n} H(\boldsymbol{Y} | R_m), \tag{5.4.2}$$

  - 其中 H($\boldsymbol{Y}$) 称为根节点的信息熵以及

$$H(\boldsymbol{Y} | R_m) = -\sum_{k=1}^{K} \hat{P}_{mk} \log_2 \hat{P}_{mk}. \tag{5.4.3}$$

  - 信息增益是自变量划分前的信息熵与自变量划分后加权信息熵的差值, 即

信息增益 = 信息熵 − 条件熵.

- 信息增益越大, 则意味着使用自变量来进行划分所获得的“纯度提升”越大.

5.4.2 信息增益

- 下面以打网球实例来演示信息增益的具体计算过程, 该实例是根据天气状况来决定是否打网球 (play tennis), 影响打网球的自变量包括天气 (weather)、温度 (temperature)、是否有风 (windy), 数据集如表 5.1 所示, 数据集中一共包括 10 个样本, 构建决策树算法对是否适合打网球进行预测.

序号	天气	温度	是否有风	是否打网球
1	晴	热	否	否
2	晴	热	是	否
3	阴	热	否	是
4	雨	温	否	是
5	雨	凉	是	否
6	雨	凉	否	是
7	阴	凉	是	是
8	晴	温	否	否
9	晴	凉	否	是
10	雨	温	否	是

5.4.2 信息增益

- 首先计算划分前根节点的信息熵, 根节点是否打网球两类的样本所占比例分别为$\frac{6}{10}$和$\frac{4}{10}$, 信息熵计算公式为

$$H(\boldsymbol{Y})=-\sum_{k=1}^{K}\hat{P}_k\log_2\hat{P}_k=-\left(\frac{6}{10}\log_2\frac{6}{10}+\frac{4}{10}\mathrm{og}_2\frac{4}{10}\right)=0.971.$$

  - 假设因变量 Y “是否打球”分别用 1 和 2 表示, $Y=1$ 表示打球, $Y=2$ 表示不打球. 在此基础上, 首先计算天气的信息增益, 对于天气, 自变量取值包括晴、阴、雨, 分别对应节点 R_1、R_2、R_3. 其中对于“晴”取值, 共有 4 个样本, 即 $n_1=4$, 打球数量占比为 $\hat{P}_{11}=\frac{1}{n_1}\sum_{i\in R_i}I(Y_i=1)=\frac{1}{4}$, 不打球数量占比为 $\hat{P}_{12}=\frac{1}{n_1}\sum_{i\in R_i}I(Y_i=2)=\frac{3}{4}$, 即天气的其他取值采用同样的计算方式. 首先根据公式(5.4.3), 用 1、2、3 分别代表晴天、阴天和雨天, 分别计算不同自变量取值的信息熵:

$$H(\boldsymbol{Y}|R_1)=-\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right)=0.811,$$

$$H(\boldsymbol{Y}|R_2)=-\left(\frac{2}{2}\log_2\frac{2}{2}\right)=0,$$

$$H(\boldsymbol{Y}|R_3)=-\left(\frac{3}{4}\log_2\frac{3}{4}+\frac{1}{4}\log_2\frac{1}{4}\right)=0.811,$$

5.4.2 信息增益

- 信息增益：

$$\Delta_{\text{weather}} = H(\boldsymbol{Y}) - \sum_{m=1}^{M} \frac{n_m}{n} H(\boldsymbol{Y}|R_m),$$

$$= 0.971 - \left(\frac{4}{10} \times 0.811 + \frac{2}{10} \times 0 + \frac{4}{10} \times 0.811 \right)$$

$$= 0.971 - 0.649 = 0.322.$$

- 同样可以求出温度和是否有风两个自变量的信息增益：

$$\Delta_{\text{temperature}} = 0.971 - \left(\frac{3}{10} \times 0.918 + \frac{4}{10} \times 0.811 + \frac{3}{10} \times 0.918 \right)$$

$$= 0.971 - 0.875 = 0.05,$$

$$\Delta_{\text{windy}} = 0.971 - \left(\frac{3}{10} \times 0.918 + \frac{7}{10} \times 0.863 \right)$$

$$= 0.971 - 0.880 = 0.09.$$

- 由信息增益的计算结果可以看出，天气的信息增益最大，因此首先选择天气作为划分自变量.

5.4.3 增益率

- 不同的离散自变量的离散值的取值个数是不同的, 即自变量不同 M 不同. 如果选取的某一自变量取值个数较多, 即 M 较大, 那么这一划分的信息增益越大. 特别地, 对于某一自变量, 如果该自变量的每个取值内只含一个样本点, 那么选取该自变量划分后的加权信息熵是 0, 进而此划分的信息增益最大. 因此, 在以信息增益作为划分训练数据集的特征时, 我们会偏向于选择取值较多的自变量. 故而, 我们引入信息增益率是为了避免自变量值个数对信息增益的影响.
- 信息增益率 (information gain ratio) 在信息增益的基础上增加了惩罚项,惩罚项是特征的固有值, 是避免上述情况而设计的. 信息增益率的定义如下:

$$\mathrm{GainR} = \frac{\Delta}{\mathrm{IV}},$$

  - 其中

$$\mathrm{IV} = -\sum_{m=1}^{M} \frac{n_m}{n} \log_2 \frac{n_m}{n}.$$

5.4.3 增益率

- 一般而言, 某个自变量的取值数目越多, IV 值越大.

- 以上面打网球的例子, 计算各自变量增益率. 首先计算各自变量的 IV 值:

$$\mathrm{IV}_{\mathrm{weather}} = -\frac{4}{10}\times\log_2\frac{4}{10} - \frac{2}{10}\times\log_2\frac{2}{10} - \frac{4}{10}\times\log_2\frac{4}{10} = 1.52,$$

$$\mathrm{IV}_{\mathrm{temperature}} = -\frac{3}{10}\times\log_2\frac{3}{10} - \frac{3}{10}\times\log_2\frac{3}{10} - \frac{4}{10}\times\log_2\frac{4}{10} = 1.57,$$

$$\mathrm{IV}_{\mathrm{windy}} = -\frac{3}{10}\times\log_2\frac{3}{10} - \frac{7}{10}\times\log_2\frac{7}{10} = 0.88.$$

- 得到三个自变量的 IV 值, 由结果可知, 天气和温度的取值个数比是否有风多, 则天气和温度的 IV 值高. 接下来计算各个自变量的信息增益率:

5.4.3 增益率

$$\mathrm{GainR}_{\mathrm{weather}} = \frac{\Delta_{\mathrm{weather}}}{\mathrm{IV}_{\mathrm{weather}}} = \frac{0.322}{1.52} = 0.212,$$

$$\mathrm{GainR}_{\mathrm{temperature}} = \frac{\Delta_{\mathrm{temperaturer}}}{\mathrm{IV}_{\mathrm{temperature}}} = \frac{0.05}{1.57} = 0.032,$$

$$\mathrm{GainR}_{\mathrm{windy}} = \frac{\Delta_{\mathrm{windy}}}{\mathrm{IV}_{\mathrm{windy}}} = \frac{0.09}{0.88} = 0.102.$$

- 求得三个自变量的增益率, 发现天气的增益率较高, 因此首先选择天气作为划分自变量.

5.4.4 基尼指数

- 对于分类因变量样本集合 $\boldsymbol{Y}$ 来说, 假定当前样本集合 $\boldsymbol{Y}$ 中第 k 类样本所占的比例为 $\hat{P}_k$ $(k=1,2,\cdots,K)$, 数据集的纯度可用基尼值 (Gini) 来度量:

$$\mathrm{Gnin}(\boldsymbol{Y})=1-\sum_{k=1}^{K}\hat{P}_k^2. \tag{5.4.4}$$

- Gini($\boldsymbol{Y}$) 反映了从数据集 $\boldsymbol{Y}$ 中随机抽取两个样本, 其类别标记不一致的概率. Gini($\boldsymbol{Y}$) 越小, 数据集 $\boldsymbol{Y}$ 的纯度越高. 基于某个离散自变量划分出的第 m 区域上的基尼值表示为

- 基于分类因变量的基尼指数 (Gini index) 定义为

$$\mathrm{GninI}=\sum_{m=1}^{M}\frac{n_m}{n}\mathrm{Gini}(\boldsymbol{Y}|R_m). \tag{5.4.5}$$

  - 基尼指数越小, 不纯度越低, 自变量越好.

5.4.5 分类误差

- 令节点 R_m 中类 k 的观测比例为， $\hat{P}_{mk}=\frac{1}{n_m}\sum_{i\in R_m} I(Y_i=k)$, 并且我们得到节点 R_m 中的所有样本的预测类别为

$$\hat{k}_m=\arg\max_k \hat{P}_{mk},$$

  - 它是节点 R_m 上样本数最多的类. 节点 R_m 上的分类误差表示为

$$\mathrm{CE}(\boldsymbol{Y}|R_m)=\frac{1}{n_m}\sum_{i\in R_m} I\left(Y\neq\hat{K}_m\right)=1-\hat{P}_{m\hat{k}_m}.$$

  - 最终也可以采用加权的思想, 得到基于 M 个节点的分类误差 (CE)

$$\mathrm{CE}=\sum_{m=1}^{M}\frac{n_m}{n}\mathrm{CE}(\boldsymbol{Y}|R_m). \tag{5.4.6}$$

5.4.6 分类误差

- 如果分裂变量 X_j 是连续的自变量以及因变量 Y 也是连续变量, 那么我们使用分裂点 t 去分裂自变量 X_j 的样本集 $\boldsymbol{X}_j = (X_{1j}, \cdots, X_{nj})^{\mathrm{T}}$. 若考虑二分裂, 使用分裂变量 X_j 和 t 可以定义两个子区域:

$$R_1(j,t) = \left\{ i \middle| X_{ij} < t \right\}, \quad R_2(j,t) = \left\{ i \middle| X_{ij} \geq t \right\},$$

  - 其中 X_{ij} 表示样本点 X_i 的第 j 个特征.

- 对应分裂变量 X_j 和分裂点 t 的均方误差定义为

$$\mathrm{MSE}(j,t) = \frac{1}{n_1} \sum_{i \in R_1(j,t)} (Y_i - \hat{c}_1)^2 + \frac{1}{n_2} \sum_{i' \in R_2(j,t)} (Y_{i'} - \hat{c}_2)^2.$$

  - 其中 $n_1 = |R_1(j, t)|$, $n_2 = |R_2(j, t)|$ 分别表示落入第一个区域和第二个区域的训练样本的数量, 且

$$\hat{c}_1 = \frac{1}{n_1} \sum_{i \in R_i(j,t)} Y_i, \quad \hat{c}_2 = \frac{1}{n_2} \sum_{i' \in R_2(j,t)} Y_{i'}.$$

5.4.7 算法总结

- 下面介绍三种决策树算法, 其中 ID3 与 C4.5 算法主要用于分类树, 而CART 算法既可以用于分类树也可以用于回归树. 对决策树 $\mathcal{T}$ 的优劣衡量可以用以下代价函数:

$$C(\mathcal{T})=\sum_{t=1}^{T} n_t H(t),$$

  - T 表示所有叶子节点的个数, n_t 是每个叶子节点中样本的个数, 在这个公式中相当于权重, 因为各叶子节点包含的样本数目不同, 可使用样本数加权求熵和. $H(t)$ 是每个叶子节点的损失度量准则, 上面章节中已介绍. 代价函数越小越好. 对所有叶子节点的熵求和, 该值越小说明对样本的分类或者回归越精确.

1. ID3 算法

- ID3 算法是一种贪心算法, 构造的决策树用于分类. 1986 年由 Quinlan[69]提出的 ID3 决策树学习算法就以信息增益为准则来选择划分自变量. ID3 算法起源于概念学习系统 (CLS), 以信息熵的下降速度为选取自变量的标准, 即在每个节点选取还尚未被用来划分的具有最高信息增益的自变量作为划分标准, 然后继续这个过程, 直到生成的决策树能完美分类训练样例.

5.4.7 算法总结

- 上述信息增益的定义是基于离散自变量的, 对于连续自变量, 我们可以采用式 (5.4.7) 给出的策略, 将连续自变量进行离散化, 即使用分裂点对连续自变量进行分裂产生区域 (即节点). 此时使用的信息增益准则, 不仅要选择自变量还要估计出分裂点.

2. C4.5 算法

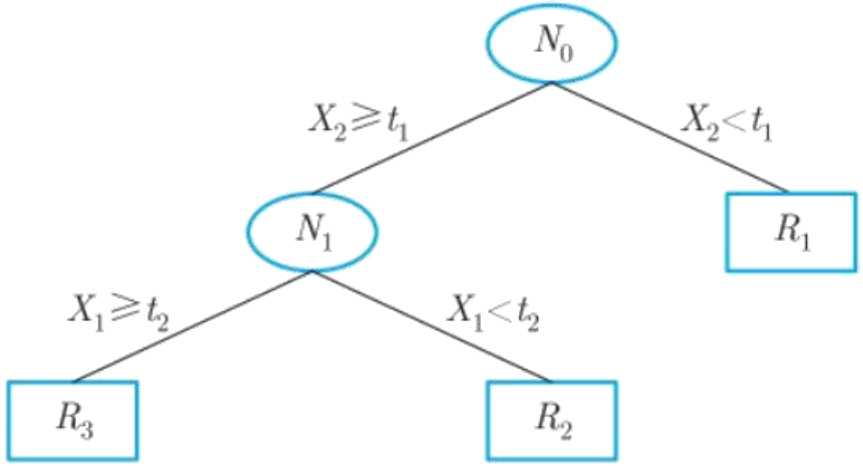
- C4.5 算法采用信息增益率来选择最优划分自变量. 不是直接选择信息增益率最大的候选划分自变量, 而是先从候选划分自变量中找出信息增益高于平均水平的自变量, 再从中选择信息增益率最高的. 采用了信息增益率来选择特征, 以减少信息增益容易选择分类取值多的自变量的问题.
- 同样的, 对于连续自变量, 我们可以采用式 (5.4.7) 给出的策略使用分裂点对连续自变量进行分裂, 然后使用上述的信息增益率准则, 选择自变量以及最优分裂点.

5.4.7 算法总结

2. CART 算法

- CART 算法 (**回归：均方误差; 分类：基尼指数**) 包括 CART 分类树和CART 回归树.

- **CART** 回归树是决策树模型的一种, 专门针对因变量是连续型随机变量或向量, 自变量是实值随机向量的树的模型. 为了便于理解, 我们首先考虑一维连续随机变量 Y , 二元自变量 X_1,X_2 的回归问题, 并以图 5.3 为例来阐述基于树的回归方法.



5.4.7 算法总结

- 首先, 按 $X_2 \geqslant t_1$ 和 $X_2 < t_1$ 把整个样本空间 N_0 划分为两个子区域 R_1和 N_1, 为了得到更加精确的预测, 可以继续对 N_1 子区域进行划分, 得到两个子区域 R_2 和R_3 . 接下来我们形象地定义一些树中的概念. 为了记号方便,我们称图 5.3 用于分裂的变量 X_2 和 X_1 为分裂变量, t_1 和 t_2 为分裂点, N_0是根节点 (包含所有样本), N_1 是中间节点 (也是 R_2 和 R_3 的父节点), R_1, R_2和 R_3 为叶子节点. 回归树需要解决以下 3 个关键问题:
  - (1) 如何选取分裂自变量? 图 5.3 第一次选择的分裂特征为 X_2, 第二次选择的分裂特征为X_1;
  - (2) 如何确定分裂点的值? 如图 5.3 中的 t_1, t_2;
  - (3) 如何确定停止分裂的准则, 即为什么叶子节点 R_1, R_2, R_3 不再分裂?
- 由问题 (1)—(3) 可知, 基于回归树模型的核心就是要依次找出分裂变量和分裂点, 制定停止规则和对每一个叶子节点的样本进行预测.
- 接下来, 我们将介绍如何逐步生成回归树. 不失一般性, 假设含有 n 个观测数据的数据集为 $D = \{(\boldsymbol{X}_1, \boldsymbol{Y}_1), \cdots, (\boldsymbol{X}_n, \boldsymbol{Y}_n)\}$, 其中, $\boldsymbol{X}_i = (X_{i1}, \cdots, X_{ip})^{\mathrm{T}}$ 是 p 维实值随机向量, Y_i 是一维实值随机变量.

5.4.7 算法总结

- 具体的做法是：从所有数据出发, 考虑分裂变量 $\boldsymbol{X}_j$ 和划分点 t 以及对应的划分成的两个子区域 $R_1(j, t)$ 和 $R_2(j, t)$, 根据表达式 (5.4.8), 通过最小化 $R_1(j, t)$ 和$R_2(j, t)$的均方误差之和

$$\min_{j,t} \mathrm{MSE}\left(j, t\right)$$

  - 得到所对应的划分变量 j 和划分点 t. 即通过搜索所有的自变量 X_j , j =1, 2, · · · , p, 以及对照分裂点 t 搜索所有的观测值 $\{X_{1j}, \cdots, X_{nj}\}$, 可以确定最好的 (j, t), 从而可以把整个样本分到两个区域. 对每一个区域重复以上过程进行分裂. 最后对所有的区域重复这一过程, 直到达到最终的停止条件, 关于停止条件及树修剪等方面详细知识可以参考文献 [67,70,71].

- CART 分类树类似回归树, 我们只需把上式中的度量标准 MSE(j, t) 替换成基尼指数, 序贯的选择最优变量去分裂样本, 从而产生节点, 最终生成分类树.

5.5　剪枝

5.5 剪 枝

- 决策树对训练数据有很好的分类和回归能力, 虽然能在训练集中取得良好的预测效果, 但对未知的测试数据未必有好的预测能力, 即泛化能力弱, 可能发生过拟合现象, 导致在测试集上效果不佳. 所以, 为了防止决策树过度生长、出现过拟合现象, 解决方案是对决策树进行剪枝或构建随机森林. 在此处主要介绍剪枝, 随机森林在后续章节进行介绍.
- 剪枝是决策树算法缓解过拟合的一种重要方法. 可通过剪枝在一定程度上避免因决策分支过多, 以致于把训练集自身的一些特点当作所有数据都具有的一般性质而导致的过拟合. 剪枝主要分为两种：预剪枝 (pre-prune) 和后剪枝 (post-prune).

5.5.1 预剪枝

- 预剪枝是在决策树构造时就进行剪枝. 方法是在构造的过程中对节点进行评估, 包括以下几种策略.
  - (1) 如果对某个节点进行划分, 在验证集中不能带来准确性的提升, 那么对这个节点进行划分就没有意义, 这时就会把当前节点作为叶子节点, 不对其进行划分.
  - (2) 或者更为直接, 预先设置每一个节点所包含的最小样本数目, 例如 10,若该节点总样本数小于 10, 则不再分;
  - (3) 或者预先指定树的高度或者深度, 例如树的最大深度为 4; 或者指定节点的熵小于某个值时就不再继续划分.

- 预剪枝的优点是可以降低训练模型的资源开销, 缺点是存在欠拟合风险.这是因为有些分支的当前划分虽然不能提升性能, 但在其基础上进行的后续划分却有可能导致性能显著提高. 预剪枝提前把这些分支剪掉了, 可能带来欠拟合风险.

5.5.2 后剪枝

- 预剪枝方法在建树过程中要求每个节点的分裂使得不纯度下降超过一定阈值. 这种方法具有一定的短视, 因为很有可能某一节点分裂不纯度的下降没超过阈值, 但是在其后续节点分裂时不纯度会下降很多, 而预剪枝法则在前一节点就已经停止分裂了.
- 后剪枝是在生成决策树之后再进行剪枝, 通常会从决策树的叶子节点开始, 逐层向上对每个节点进行评估. 如果剪掉这个节点子树, 与保留该节点子树在分类准确性上差别不大, 或者剪掉该节点子树, 能在验证集中带来准确性的提升, 那么就可以把该节点子树进行剪枝.
- 后剪枝的优点是比预剪枝保留了更多的分支, 欠拟合风险小, 泛化性能往往优于预剪枝决策树, 缺点是模型训练时会占用较多的资源.
- 在实际应用中, 更多的是使用后剪枝法, 其中的代价复杂性剪枝 (costcomplexity pruning) 是最常用的方法. 这种方法是先让树尽情生长, 得到 $\mathcal{T}_0$,然后再在 $\mathcal{T}_0$ 基础上进行修剪.

5.5.2 后剪枝

- 设 T 表示子树 $\mathcal{T}$ 的叶子节点数目, n_t 表示叶子节点 t 的样本量, 则代价复杂性剪枝法的损失函数为

$$C_\alpha(\mathcal{T}) = \sum_t^T n_t H(t) + \alpha T, \tag{5.5.1}$$

  - 其中 $\sum_{t=1}^{T} n_t H(t)$ 是代价函数, α 是参数. $\mathcal{T}$ 是任意一棵子树, 它通过对 $\mathcal{T}_0$ 进行剪枝得到, 也就是减去 $\mathcal{T}_0$ 某个中间节点的所有子节点, 使其成为 $\mathcal{T}$ 的叶子节点. 对于固定的 α, 一定存在这样一棵子树 $\mathcal{T}_\alpha$ 使得损失函数 $C_\alpha(\mathcal{T})$ 达到最小.

- 调整参数 α 控制着模型对数据的拟合与模型的复杂度 (树的大小) 之间的平衡. 当 $\alpha = 0$ 时, 最优子树 $\mathcal{T}_\alpha$ 等于原树 $\mathcal{T}_0$. 随着 α 取值增大, 损失函数对叶子节点数目 T 惩罚增大, 那么此时我们偏向于选择叶子节点数目少一点的树.

- 后剪枝步骤: 从 $\mathcal{T}_0$ 开始, 自下而上地考虑每个内部节点 t , 考虑两种情况:

  - (1) 以 t 为根节点的子树 $\mathcal{T}_t$, 其损失为 $C_\alpha(\mathcal{T}_t) = C(\mathcal{T}_t) + \alpha T_t$;
  - (2) 对 t 进行剪枝, 即将 $\mathcal{T}_t$ 作为叶子节点, 其损失为 $C_\alpha(t) = C(t) + \alpha$;

5.5.2 后剪枝

- 我们通过比较 $C_\alpha(T_t)$ 与 $C_\alpha(t)$ 大小来判断是否要对 t 进行剪枝:
  - 当 α 较小时, 可以容忍模型有较高的复杂度, 所以 $C_\alpha(T_t) < C_\alpha(t)$, 即这个时候不需要剪枝;
  - 当 α 逐渐增大到某一阈值时, 这个时候需要考虑在训练数据上 T 的损失增大, 所以有 $C_\alpha(T_t) = C_\alpha(t)$, 即这个时候剪不剪枝都可以;
  - 当 α 继续增大, 以至于大于上述阈值时, 有 $C_\alpha(T_t) > C_\alpha(t)$, 这个时候剪枝带来的收益大于作为一棵子树 T_t 所带来的收益, 所以要剪枝.
- 从以上过程可以看出, 对于树中的每个内部节点 t, 都有一个特定的阈值$g(t)$. 当 $\alpha = g(t)$ 时, $C_\alpha(T_t) = C_\alpha(t)$. 故而 $g(t)$ 可以决定是否需要对其进行剪枝, 且该阈值等于

$$g(t) = \frac{C(t) - C(T_t)}{T_t - 1}.$$

  - 上式 $g(t)$ 可由等式 $C_\alpha(T_t), C_\alpha(t)$ 解出.

5.5.2 后剪枝

- 因此, 在生成子树 $\mathcal{T}_1$ 时, 我们可以计算 $\mathcal{T}_0$ 的每个内部节点的阈值 $g(t)$,选择其中最小的记为 $g(t_1)$. 当 α 大于该阈值 $g(t_1)$ 时, 这意味着要剪掉该节点 t_1 对应的子树 $\mathcal{T}_t$ 得到新的树 $\mathcal{T}_1$ 收益较大. 当 $g(t_1) \leq \alpha < g(t_2)$ 时, $\mathcal{T}_1$ 使得损失函数最小. 同理当 $g(t_2) < \alpha$ 时, 剪掉节点 t_2 的子树, 使 t_2 为叶子节点, 得到子树 $\mathcal{T}_2$. 同理子树 $\mathcal{T}_2$ 在区间 $g(t_2) \leq \alpha < g(t_3)$ 上使得损失函数最小. 接着在 $\mathcal{T}_2$的基础上持续剪枝, 就可以得到最终的子树序列.
- 在生成子树序列 $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \cdots$ 后, 使用验证数据集进行交叉验证, 测试子树序列中每棵子树的损失, 选择最小的子树作为剪枝后的决策树, 这个时候也对应了一个 α_k .

5.6　决策树实践

实践代码