

机器学习与深度学习 ——决策树 (Decision Tree)



Personal Website: <https://www.miaopeng.info/>



Email: miaopeng@stu.scu.edu.cn



Github: <https://github.com/MMeowhite>



Youtube: <https://www.youtube.com/@pengmiao-bmm>

目录章节

CONTENTS

01 决策树的概念与原理

02 模型求解与评估

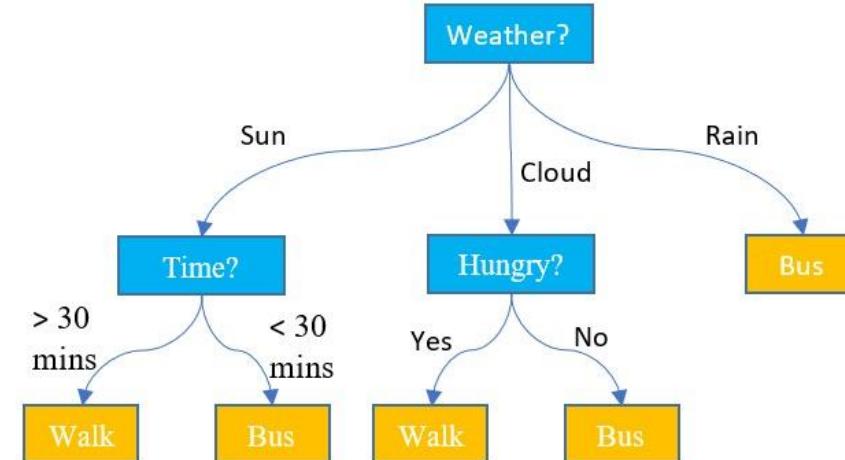
03 模型扩展与改进

04 模型实现与实战

05 总结

▶ 为什么学决策树？

- 决策树模型直观、可解释性强，能像if-else规则一样解释特征与标签的关系。决策树通过层层划分特征空间，能够捕捉数据的非线性关系和特征之间的交互作用。
- 它既是独立的高效模型，也作为集成学习（随机森林、Boosting）的核心基元，在工业界和学术界广泛应用。

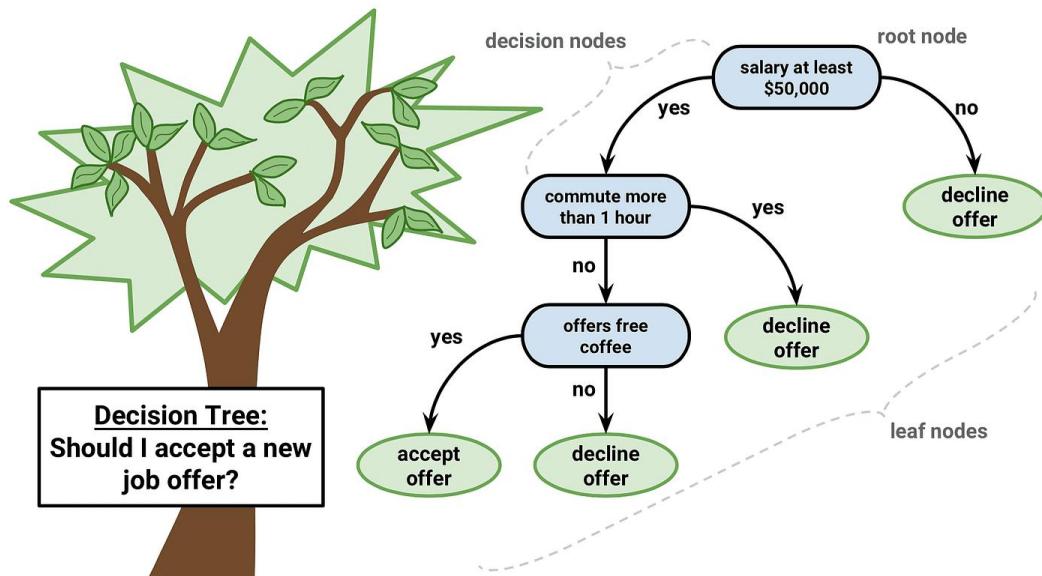


- 决策树的结构就像一颗倒着长的树：根->主干->支干->叶子

决策树是机器学习的第五个台阶，代表了从“线性假设”到“非线性分而治之”的跃升。

▶ 什么是决策树？

- 决策树是一种通过“逐步提问并分裂数据”来做出分类或预测的树状模型，自上而下在特征空间中递归划分，在每个内部节点以信息增益/基尼指数/方差等准则选择最优划分，最终由叶节点以多数投票（分类）或目标均值/常数（回归）给出预测。
- 本质：基于特征递归划分空间的条件规则集合。



- **根节点 (Root)**：就像树的根部，所有决策从这里开始。比如“天气”是不是晴天。
- **分支 (Branch)**：像树的枝干，一条规则把问题划分成不同情况，比如“下雨 → 打伞”，“晴天 → 不打伞”。
- **叶子节点 (Leaf)**：像树的叶子，表示最终的决策结果，例如“出去玩”或者“待在家”。
- **生长过程**：树从根长出枝干、枝干再分叉，就像决策树不断用特征把样本划分得越来越细。

► 决策树的前置知识：熵与信息量

➤ 1) 信息量 (Information Content)

- 直观理解：信息量衡量的是一个事件发生时带来的“惊讶程度”。事件越稀少/不常见，发生时信息量越大；事件越常见，信息量越小。
- 数学定义：对于事件x发生的概率 $p(x)$ ，信息量定义为：

$$I(x) = -\log_2 p(x)$$

单位：比特(bit)

- 举例：如果概率 $p=1/2$ ，则 $I(x)=-\log_2(1/2)=1\text{bit}$ ；如果概率 $p=1/4$ ，则 $I(x)=-\log_2(1/4) = 2\text{bit}$ 。

➤ 2) 熵 (Entropy)

- 直观理解：熵是随机变量不确定性的整体度量，也就是平均信息量。数据越混乱、不确定性越高->熵越大；数据越纯、确定性越高->熵越小。
- 数学定义：对于一个离散随机变量X可能取值 x_1, x_2, \dots, x_n ，熵定义为：

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

- 举例：假设数据集只有两类A和B，且比例均等 $P(A)=P(B)=0.5$ ， $H=-(1/2\log_2(1/2)+1/2\log_2(1/2))=1$ ，代表最大确定性（完全混合）；如果所有样本都是A，即 $P(A)=1$ ， $P(B)=0$ ， $H=-(\log_2(1)+0\log_2(0))=0$ ，代表完全确定，没有不确定性。

▶ 决策树的前置知识：信息增益

➤ 3) 信息增益 (Information Gain)

- 直观理解：划分数据后，样本的不确定性（熵）减少了多少。如果 $IG > 0$ ，代表熵减少了，说明信息更加确定化，反之如果 $IG < 0$ ，代表熵增，信息更加模糊、不确定化。常用在ID3分类树。
- 数学定义：假设 $Ent(D)$ 为划分前的熵， $Ent(D_v)$ 为划分后每个子集的熵， $|D_v|$ 代表子集样本数：

$$IG(D, A) = Ent(D) - \sum_{v=1}^M \frac{|D_v|}{|D|} Ent(D_v)$$

信息增益越大，说明用特征A划分后越“纯”，分类效果越好。

- 举例：预测今天是否打球（是/否），特征有天气（晴天、阴天、雨天）

天气	打球
晴天	否
晴天	否
阴天	是
雨天	是
雨天	否
晴天	是
阴天	是
雨天	是
晴天	是
阴天	是

✓ 计算总体熵，目标变量“打球”有是=6，，否=4，那么总体熵有：

$$Ent(D) = -\frac{6}{10} \log_2 \frac{6}{10} - \frac{4}{10} \log_2 \frac{4}{10} \approx 0.971$$

✓ 按照“天气”分组：

$$Ent(\text{sunny}) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

$$Ent(\text{overcast}) = -1 \log_2 1 - 0 \log_2 0 = 0$$

$$Ent(\text{rainy}) = -\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} \approx 0.918$$

✓ 加权平均熵：

$$Ent(\text{weather}) = \frac{4}{10} \cdot 1 + \frac{3}{10} \cdot 0 + \frac{3}{10} \cdot 0.918 \approx 0.675$$

✓ 计算信息增益GI

$$IG(D, \text{weather}) = Ent(D) - Ent(\text{weather}) = 0.971 - 0.675 = 0.296$$

▶ 决策树的前置知识：熵与信息量

➤ 4) 基尼指数 (Gini Index)

- 直观理解：随机从集合D中抽两个样本，它们属于不同类别的概率。常用在CART分类树。
- 数学定义：对于一个离散随机变量X可能取值 x_1, x_2, \dots, x_n ，熵定义为：

$$Gini(D) = 1 - \sum_{k=1}^K p_k^2$$

基尼指数通常是指**目标变量的混杂程度**，因此必须结合 目标变量的分布来计算。

- 举例：预测今天是否打球（是/否），特征有天气（晴天、阴天、雨天）

天气	打球
晴天	否
晴天	否
阴天	是
雨天	是
雨天	否
晴天	是
阴天	是
雨天	是
晴天	是
阴天	是

✓ 总样本=10，其中针对打球与否，分为两类，是=7，否=3，即：

$$p(Yes) = 0.7, \quad p(No) = 0.3$$

✓ 所以基尼指数为：

$$Gini(D) = 1 - (0.7^2 + 0.3^2) = 1 - (0.49 + 0.09) = 0.42$$

► 决策树的前置知识：方差

► 4) 方差 (Variance)

- 直观理解：方差衡量数据点与均值的偏离程度。小方差→样本值接近，预测更稳定；大方差 → 样本值分散，预测误差大。
- 数学定义：

$$\text{Var} = \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \bar{y})^2, \quad \bar{y} = \frac{1}{|D|} \sum_{i=1}^{|D|} y_i$$

- 在回归树中的思路：在 分类树 中，我们希望每个节点里的样本尽量属于同一类（低不纯度），在 回归树 中，我们希望每个节点里的样本 y_i 尽量接近一个常数（通常是该节点的均值）
- 举例：假设我们想用房子的 面积（平方米） 来预测 房价（万元）。

✓ 计算整体的方差：

面积	房价
50	150
60	160
70	170
90	220
100	230
110	240

$$\bar{y} = \frac{150 + 160 + 170 + 220 + 230 + 240}{6} = 195, \quad \text{Var}(D) = \frac{(150 - 195)^2 + (160 - 195)^2 + \dots + (240 - 195)^2}{6} = 1166.7$$

✓ 尝试划分：我们考虑按 面积 来划分，比如在 80 平米处分开：a.左子集 (≤ 80) : {150, 160, 170}, 均值 = 160, 方差 = 66.7; b.右子集 (> 80) : {220, 230, 240}, 均值 = 230, 方差 = 66.7

$$\text{VarSplit} = \frac{3}{6} \times 66.7 + \frac{3}{6} \times 66.7 = 66.7$$

✓ 对比整体方差 1166.7 → 下降非常多，说明划分点 = 80 很好！

► 决策树的数学表达形式

- 假设训练集为 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, $x_i \in \mathbb{R}^d$, $y_i \in Y$ 。
 - 1) 节点划分: 决策树通过在某个特征 A_j 上选择一个划分点 t , 将数据集 D 划分为:
$$D_{left} = \{(x, y) \in D | x_j \leq t\}, \quad D_{right} = \{(x, y) \in D | x_j > t\}$$
 - 对于分类问题(特征可以是离散/连续), 离散特征则按照取值划分。
 - 2) 划分准则: 根据分类任务和回归任务分为两类: 分类任务主要利用常用信息增益或基尼指数; 回归任务: 常用方差最小化。
 - 3) 递归构建: 决策树采用**递归分裂策略**:

$$\text{Split}(D) = \arg \max_{A, t} \Delta(D, A, t)$$

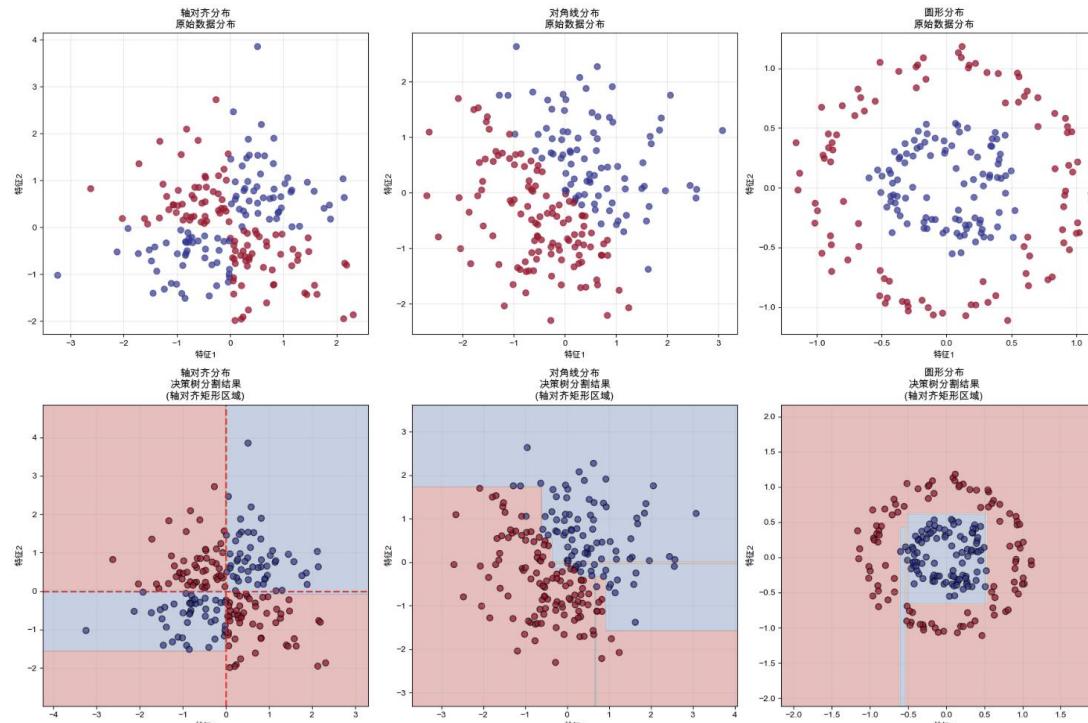
 - 其中 Δ 表示划分增益(信息增益、基尼指数减少量、方差减少量)。
 - 停止条件: 节点样本数小于阈值、节点样本全属同一类、达到最大树深。
 - 4) 预测函数:
 - 分类树: 对于叶节点 L , 预测为多数类: $\hat{y} = \arg \max_k p_k$
 - 回归树: 对于叶子点 L , 预测为叶节点样本的均值: $\hat{y} = \frac{1}{|L|} \sum_{(x_i, y_i) \in L} y_i$

决策树通过递归选择最优特征划分来最小化节点不纯度, 直到满足停止条件; 最终预测由叶节点给出, 后续还可以通过剪枝避免过拟合。

► 模型假设（非常重要！）

- 决策树的模型同样隐含了一些重要假设，如果这些假设在数据中不成立，那么模型的拟合效果往往显著下降。

- 1) 特征空间可被递归地划分成若干个区域

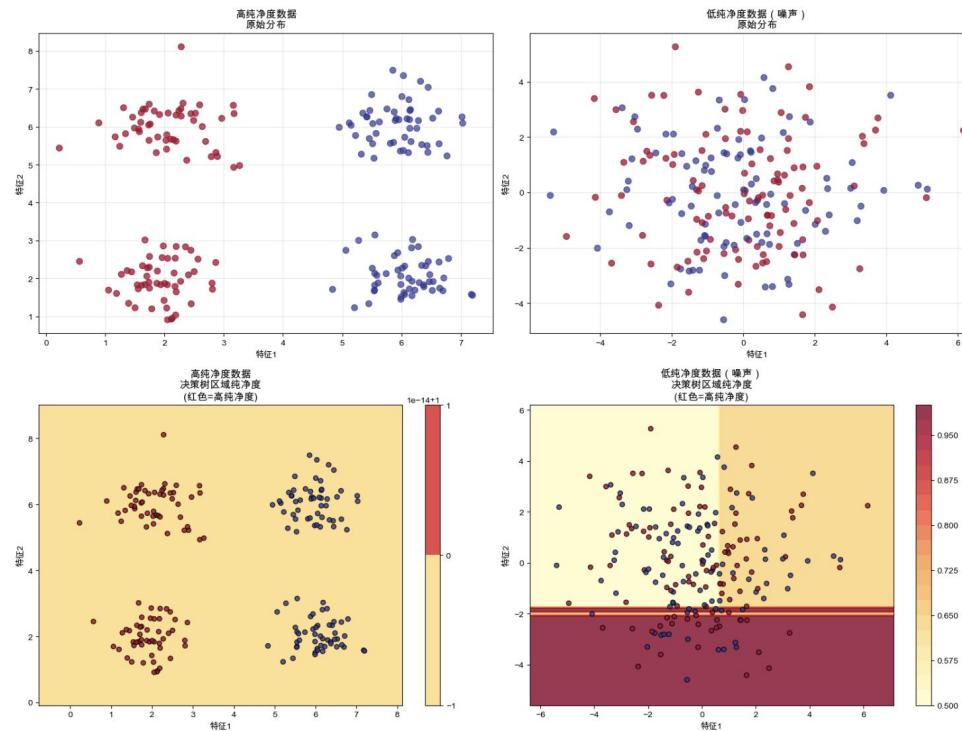


- 假设数据分布可以通过一系列“逐步二分（或多分）”操作来切分。
 - 每次划分都是单一特征上的阈值比较（轴对齐的切分，而不是任意方向的超平面）。

► 模型假设（非常重要！）

- 决策树的模型同样隐含了一些重要假设，如果这些假设在数据中不成立，那么模型的拟合效果往往显著下降。

- 2) 每个区域内部是相对“纯净”的

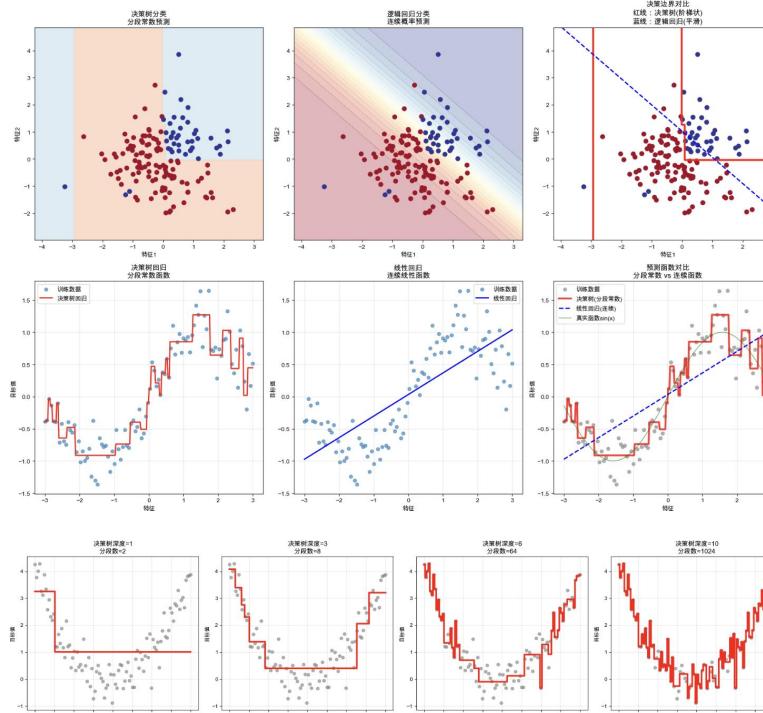


- 分类树：假设区域内样本属于同一类别（或者大部分属于同一类别）。
 - 回归树：假设区域内目标值接近一个常数（通常用均值表示）。

► 模型假设（非常重要！）

- 决策树的模型同样隐含了一些重要假设，如果这些假设在数据中不成立，那么模型的拟合效果往往显著下降。

- 3) 预测方式是分区常数模型



- 决策树本质上是一个 piecewise constant function（分段常数函数），类似于Sign函数。即：输入空间被划分为多个小矩形区域，每个区域给出一个固定预测值（多数类别 / 平均值）。

► 小结

- 为什么需要学习决策树?
 - 学习 SVM 能帮助我们理解最大间隔原理与核方法，掌握一种在小样本、高维度下仍具强大泛化能力的经典机器学习模型。
- 什么是KNN（K-近邻算法）?
 - SVM（支持向量机）是一种通过寻找最大化分类间隔的超平面来实现二分类的机器学习模型。它可通过核函数将数据映射到高维空间，从而解决线性不可分问题，兼具良好的泛化能力。
 - SVM的代价函数：
$$\min \frac{1}{2} \|w\|_2^2 \quad s.t. \quad y_i(w^T x_i + b) \geq 1 \quad (i = 1, 2, \dots, m)$$
- 模型假设:
 - 特征空间可被递归地划分成若干个区域。
 - 每个区域内部是相对“纯净”的。
 - 预测方式是分区常数模型。

学习 SVM 可以掌握一种基于最大间隔的判别式算法，它通过最优超平面与核函数实现分类预测，假设数据在合适空间中近似可分。

目录章节

CONTENTS

01

决策树的概念和原理

02

模型求解与评估

03

模型扩展与改进

04

模型实现与实战

05

总结

► 模型求解：概述

- 决策树的求解就是**如何找到最优划分**，典型步骤是**自顶向下递归生成**：
 - 1) 选择划分特征和阈值：对每个候选特征，尝试不同划分点。使用不同的指标进行最佳划分：分类树——>信息增益、信息增益率、基尼指数；回归树——方差最小化。常用算法如下：
 - ID3 (Iterative Dichotomiser 3)：划分准则为信息增益，特点：选择信息增益最大的特征作为划分节点，倾向于选择取值多的特征（可能过拟合）
 - C4.5 (ID3的改进)：划分准则为信息增益率，特点：解决 ID3 偏好多值特征的问题，支持连续特征划分（通过阈值），可以处理缺失值。
 - CART (Classification And Regression Tree)：划分准则为基尼系数（分类）或方差最小化（回归），特点：生成二叉树（每个节点只有两个子节点），支持分类与回归，便于剪枝。
 - 2) 划分数据集：根据选定特征和阈值，把数据集分成子集（左/右或多分支）。
 - 3) 递归构建子树：对每个子集重复第一步和第二步，直到满足停止条件：节点样本数小于最小样本数；达到最大树深；子集纯度够高（分类）或方差足够小（回归）。
 - 4) 生成叶节点预测：分类树——多数类别或类别概率；回归树——均值或常数。

► 模型求解：1) ID3

- 目标输入 $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i = (x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F = \{A_1, \dots, A_m\}$, 目标输出为决策树 T 。

■ Step1: 计算节点熵 (信息量)

- 节点数据集 D 的信息熵:

$$H(D) = - \sum_{c=1}^k p_c \log_2 p_c$$

■ Step2: 计算每个特征的信息增益

- 特征 A 有取值集合 $\{a_1, \dots, a_v\}$, 划分后的子集为 D_{aj} , 则条件熵:

$$H(D|A) = \sum_{j=1}^v \frac{|D_{aj}|}{D} H(D_{aj})$$

- 信息增益:

$$\text{Gain}(D, A) = H(D) - H(D|A)$$

■ Step3: 选择最优特征划分

$$A^* = \arg \max_{A \in F} \text{Gain}(D, A)$$

- 用 A^* 对当前节点进行划分, 划分完成之后将 A^* 从候选特征集合中移除。

► 模型求解：1) ID3

- 目标输入 $D=\{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i=(x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F=\{A_1, \dots, A_m\}$, 目标输出为决策树 T 。
- Step4: 递归构建子树
 - 对每个子集 D_{aj} , 如果 D_{aj} 全部属于同一类别 \rightarrow 生成叶节点; 如果没有剩余特征 \rightarrow 叶节点取多数类别; 否则 \rightarrow 对 D_{aj} 重复步骤1 ~ 4。
- Step5: 终止
 - 终止条件为: a. 节点样本全部属于同一类别; b. 没有可用特征; c. 样本数小于最小阈值
- ID3算法特点:

特点	描述
优点	简单直观, 易于实现
缺点	偏向多值特征, 容易过拟合
划分准则	信息增益
树类型	多叉树
适用任务	分类任务

► 模型求解：1) ID3

➤ 假设我们有如下的数据：

Weather	Temperature	Humidity	Windy	Go outside
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes

- 特征：Weather(Sunny=4, Rainy=4, Overcast=2); Temperatuer(Hot=3, Mild=3, Cool=4); Humidity(High=5, Normal=5), Windy(True/False)。目标：Go outside(Yes/No)₈

► 模型求解：1) ID3

➤ 根节点熵（整体熵）

- 首先需要求解整体熵：

$$H(D) = - \sum_{c \in Yes, No} p_c \log_2 p_c = -0.6 \log_2 0.6 - 0.4 \log_2 0.4 \approx 0.97095$$

➤ 对每个特征计算信息增益

- 根据信息增益公式

$$IG(D, A) = H(D) - \sum_{v=1}^M \frac{|D_v|}{|D|} Ent(D_v)$$

- 对于特征weather: Sunny(4, yes=1, no=3) $\rightarrow H(\text{Sunny}) = -1/4 \log_2 1/4 - 3/4 \log_2 3/4 = 0.8113$, Overcast(2, yes=2, no=0) $\rightarrow H(\text{Overcast}) = 0$, Rainy(4, yes=3, no=1) $= 0.8113$, 因此加权条件熵为：

$$H(D|Weather) = \frac{4}{10} \cdot 0.81128 + \frac{2}{10} \cdot 0 + \frac{4}{10} \cdot 0.81128 = 0.6490$$

- 因此，信息增益为：

$$IG(D, Weather) = H(D) - H(D|Weather) = 0.9710 - 0.6490 = 0.3220$$

- 同理，可以计算其他几个特征的信息增益：

$$IG(D, Temperature) = H(D) - H(D|Temperature) = 0.095$$

$$IG(D, Humidity) = H(D) - H(D|Humidity) = 0.1245$$

$$IG(D, Windy) = H(D) - H(D|Windy) = 0.091$$

► 模型求解：1) ID3

➤ 根节点的选择

- 比较信息增益，发现 $IG(D, Weather)$ 的信息增益最大，因此ID3算法在根节点选择Weather作为划分属性。

➤ 对每个子节点递归：

- 子集：Weather=Overcast，都是Yes表示能直接成为叶子节点（纯节点）
- 子集：Weather=Sunny，在Sunny子集上尝试划分(Humidity, Temperature) Temperature，如果按照Humidity划分: High (3 samples): all No \Rightarrow 熵=0; Normal (1 sample): all Yes \Rightarrow 熵=0，加权信息熵为0，所以信息增益为 $1-0=1$ ，因此 Sunny 节点选择Humidity，并且得到两个纯叶。Sunny \wedge Humidity=High \rightarrow No, Sunny \wedge Humidity=Normal \rightarrow Yes
- 子集：Weather=Rainy，在Rainy子集上尝试划分(Humidity, Temperature) Temperature，如果按照Windy划分: False(3 samples): all Yes \Rightarrow 熵=0; True (1 sample): all No \Rightarrow 熵=0，加权信息熵为0，所以信息增益为 $1-0=1$ ，因此 Rainy节点选择Windy，并且得到两个纯叶。Sunny \wedge Windy=False \rightarrow Yes, Sunny \wedge Windy = True \rightarrow No
- 决策树：Weather?
 - Overcast: Yes
 - Sunny:
 - Humidity = High : No
 - Humidity = Normal: Yes
 - Rainy:
 - Windy = False: Yes
 - Windy = True : No

➤ 小结：1) ID3 用信息增益作为划分准则：选择能最大减少熵（不确定性）的特征；2) 递归构建直到叶子纯或无特征可用（或触发停止条件）。

► 模型求解：2) C4.5

- 目标输入 $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i = (x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F = \{A_1, \dots, A_m\}$, 目标输出为决策树 T 。

- Step1: 计算节点熵（信息量）
 - 节点数据集 D 的信息熵:

$$H(D) = - \sum_{c=1}^k p_c \log_2 p_c$$

- Step2: 计算每个特征的信息增益
 - 特征 A 有取值集合 $\{a_1, \dots, a_v\}$, 划分后的子集为 D_{aj} , 则条件熵:

$$H(D|A) = \sum_{j=1}^v \frac{|D_{aj}|}{|D|} H(D_{aj})$$

- 信息增益率:

$$\begin{aligned} \text{Gain}(D, A) &= H(D) - H(D|A), & IV(A) &= - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \\ \text{GainRatio}(D, A) &= \frac{\text{Gain}(D, A)}{IV(A)} \end{aligned}$$

- 选择最优特征划分

$$A^* = \arg \max_{A \in F} \text{GainRatio}(D, A)$$

- 小技巧: C4.5 实际操作时, 先筛选出信息增益大于平均水平的特征, 再在其中选择信息增益率最高的, 避免过度偏向取值特别多的特征。

► 模型求解：2) C4.5

- 目标输入 $D=\{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i=(x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F=\{A_1, \dots, A_m\}$, 目标输出为决策树 T 。
- Step4: 递归构建子树
 - 对每个子集 D_{aj} , 如果 D_{aj} 全部属于同一类别 \rightarrow 生成叶节点; 如果没有剩余特征 \rightarrow 叶节点取多数类别; 否则 \rightarrow 对 D_{aj} 重复步骤1 ~ 4。
- Step5: 终止
 - 终止条件为: a. 节点样本全部属于同一类别; b. 没有可用特征; c. 样本数小于最小阈值
- 特点: **支持连续值/缺失值处理**
 - C4.5 支持连续特征, 方法是: 对某连续特征 A 先排序, 然后选择候选划分点 (通常是相邻样本取值的中点), 最后找到能最大信息增益率的划分点, 作为该特征的二元划分标准。
 - C4.5 也支持缺失处理: 用概率分配样本到不同分支而不是直接丢弃, 同时, 在计算信息增益率的时候按样本权重来考虑缺失。

► 模型求解：2) C4.5

➤ 根节点熵（整体熵）

- 首先需要求解整体熵：

$$H(D) = - \sum_{c \in Yes, No} p_c \log_2 p_c = -0.6 \log_2 0.6 - 0.4 \log_2 0.4 \approx 0.97095$$

➤ 对每个特征计算信息增益

- 根据信息增益公式

$$IG(D, A) = H(D) - \sum_{v=1}^M \frac{|D_v|}{|D|} Ent(D_v)$$

- 对于特征weather: Sunny(4, yes=1, no=3) $\rightarrow H(\text{Sunny}) = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4} = 0.8113$, Overcast(2, yes=2, no=0) $\rightarrow H(\text{Overcast}) = 0$, Rainy(4, yes=3, no=1) $= 0.8113$, 因此加权条件熵为：

$$H(D|Weather) = \frac{4}{10} \cdot 0.8113 + \frac{2}{10} \cdot 0 + \frac{4}{10} \cdot 0.8113 = 0.6490$$

- 因此，信息增益率为：

$$IG(D, Weather) = H(D) - H(D|Weather) = 0.9710 - 0.6490 = 0.3220$$

$$IV(Weather) = -\left(\frac{4}{10} \log_2 \frac{4}{10} + \frac{2}{10} \log_2 \frac{2}{10} + \frac{4}{10} \log_2 \frac{4}{10}\right) = 1.5219$$

$$GR(Weather) = \frac{0.3220}{1.5219} = 0.2115$$

► 模型求解：2) C4.5

- 同理，可以计算其他几个特征的信息增益率：

$$GR(Temperature) = 0.061 \quad GR(Humidity) = 0.1245 \quad GR(Windy) = 0.1036$$

➤ 根节点的选择

- 比较信息增益，发现RG(Weather)的信息增益率最大，因此C4.5算法在根节点选择Weather作为划分属性。

➤ 对每个子节点递归：

- 子集：Weather=Overcast，都是Yes表示能直接成为叶子节点（纯节点）
- 子集：Weather=Sunny (4 样本: Yes=1, No=3, 节点熵 0.8113)，计算剩余特征 (Temp / Humidity / Windy) 的 GR: 1) 针对Humidity: High 全 No→H=0, Normal全Yes→H=0 ⇒ H(D|Humidity) =0, Gain=0.8113, IV=-(3/4log₂3/4+1/4+log₂1/4)=0.8113 ⇒ GR=1; 2) 针对Temperature: Gain=0.8113, 但是IV=-(1/2log₂1/2 + 1/4log₂1/4+1/4log₂1/4) =1.5 ⇒ GR=0.5409; 3) 针对Windy: 同理可得GR=0.1511。
- 子集：Weather=Rainy (4 样本: Yes=3, No=1, 节点熵 0.8113)，计算剩余特征 (Temp / Humidity / Windy) 同理计算，GR(Windy)=1, GR(Temperature)=0.3113, GR(Humidity)=0.1511。
- 决策树：Weather?
 - └ Overcast: Yes
 - └ Sunny:
 - └ Humidity = High : No
 - └ Humidity = Normal: Yes
 - └ Rainy:
 - └ Windy = False: Yes
 - └ Windy = True : No

➤ 小结：C4.5 在 ID3 的基础上做了改进，引入信息增益率 (Gain Ratio)，在信息增益的基础上引入分裂信息 (Intrinsic Value, IV) 来惩罚取值多的特征，避免这种偏好。

► 模型求解：2) C4.5

- C4.5在ID3的基础上不仅通过信息增益率抑制多值偏好，而且可以进一步扩展到连续特征和缺失值处理，使得决策树更合理、更健壮、更适用实际场景。

- 连续特征：

Weather	Temperature	Go outside
Sunny	72	Yes
Sunny	85	No
Rainy	69	Yes
Overcast	90	Yes
Rainy	75	No

- ID3只能处理离散特征，所以对这里的“温度”没办法直接用，需要先手工离散化（比如低温、中温、高温），但不同划分方式结果差异大。
- C4.5算法：C4.5会自动找到最优切分点。步骤：先把温度排序69, 72, 75, 85, 90，然后在相邻值之间取候选切分点70, 73.5, 80, 87.5，最后分别计算信息增益率，选最优（比如“温度 ≤ 73.5 ” vs “ > 73.5 ”）。

► 模型求解：2) C4.5

- C4.5在ID3的基础上不仅通过信息增益率抑制多值偏好，而且可以进一步扩展到连续特征和缺失值处理，使得决策树更合理、更健壮、更适用实际场景。

- 缺失值处理：

Weather	Humidity	Go outside
Sunny	High	No
Sunny	?	Yes
Rainy	Normal	Yes
Overcast	High	Yes
Rainy	?	No

- ID3如何处理？ID3 要么丢掉缺失值样本（浪费数据），要么简单填充（可能带来偏差）。
- C4.5算法：C4.5 在计算信息增益率时会对缺失值样本分配权重。步骤：比如“Sunny, 湿度=?”这个样本：在“湿度=High”分支分配0.6权重，在“湿度=Normal”分支分配0.4权重（按已知样本比例）。这样缺失数据不会被丢掉，也不会过分偏向某个分支。更合理地利用数据，鲁棒性更好。

► 模型求解：3) CART

- 目标输入 $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i = (x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F = \{A_1, \dots, A_m\}$, 目标输出为决策树 T 。

- Step1: 计算根节点的基尼指数
 - 基尼指数公式

$$Gini(D) = 1 - \sum_k p_k^2$$

- Step2: 计算每个特征的信息增益
 - 特征 A 有取值集合 $\{a_1, \dots, a_v\}$, 划分后的子集为 D_{aj} , 则子集的基尼指数为:

$$Gini(D_i) = 1 - \sum_k p_{k|i}^2$$

- 基尼下降值:

$$Gini_{split}(D, A) = Gini(D) - \sum_{i=1}^v \frac{|D_i|}{|D|} Gini(D_i)$$

- Step3: 选择最优特征划分

$$A^* = \arg \max_{A \in F} Gini_{split}(D, A)$$

► 模型求解：3) CART

- 目标输入 $D=\{(x_1, y_1), \dots, (x_n, y_n)\}$, 每个样本 $x_i=(x_{i1}, \dots, x_{im})$, 标签 $y_i \in \{C_1, \dots, C_k\}$, 候选特征集合 $F=\{A_1, \dots, A_m\}$, 目标输出为决策树 T 。
- Step4: 递归构建子树
 - 对每个子集 D_{aj} , 如果 D_{aj} 全部属于同一类别 \rightarrow 生成叶节点; 如果没有剩余特征 \rightarrow 叶节点取多数类别; 否则 \rightarrow 对 D_{aj} 重复步骤1 ~ 4。
- Step5: 终止
 - 终止条件为: a. 节点样本全部属于同一类别 (基尼指数 (或方差) = 0); b. 没有可用特征; c. 样本数小于最小阈值
- CART算法特点:

特点	描述
优点	能同时处理分类和回归任务
缺点	容易过拟合, 划分标准单一, 只能生成二叉树, 对异常值和噪声敏感
划分准则	基尼指数 (分类)、最小化方差 (回归)
树类型	二叉树
适用任务	分类、回归任务均可

► 模型求解：3) CART

➤ 计算根节点基尼指数

- 这里的目标有两类 ($Yes=6$, $No=4$) , 所以 $P_{Yes}=0.6$, $P_{No}=0.4$:

$$Gini(D) = 1 - (0.6^2 + 0.4^2) = 0.48$$

➤ 对每个特征计算信息增益

- 根据加权基尼公式:

$$Gini_{split}(D, A) = Gini(D) - \sum_{i=1}^v \frac{|D_i|}{|D|} Gini(D_i)$$

- 对于特征weather: Sunny(4, yes=1, no=3), Overcast(2, yes=2, no=0), Rainy(4, yes=3, no=1), 因此各个指标的基尼指数为:

$$Gini(D, Sunny) = 1 - (0.25^2 + 0.75^2) = 1 - (0.0625 + 0.5625) = 0.375$$

$$Gini(D, Overcast) = 1 - (1^2 + 0^2) = 1 - (1 + 0) = 0$$

$$Gini(D, Rainy) = 1 - (0.75^2 + 0.25^2) = 1 - (0.5625 + 0.0625) = 0.375$$

- 因此加权基尼指数为:

$$Gini(D, Weather) = \frac{4}{10} \cdot 0.375 + \frac{2}{10} \cdot 0 + \frac{4}{10} \cdot 0.375 = 0.15 + 0 + 0.15 = 0.30$$

- 因此基尼下降值为: $Gini_{split}(D, Weather) = 0.48 - 0.30 = 0.18$

► 模型求解：3) CART

➤ 根节点的选择

- 同理可以计算其他特征的基尼下降值（这里空间太小写不下）， $Gini(D, Weather)$ 的基尼指数下降最大，因此CART算法在根节点选择Weather作为划分属性。

➤ 对每个子节点递归：

- 子集：Weather=Overcast($Gini=0$)，都是Yes表示能直接成为叶子节点（纯节点），无需继续划分。
- 子集：Weather=Sunny($Gini=0.375$, samples=4)，在Sunny子集上尝试划分（Humidity, Temperature），如果按照Humidity \Rightarrow High(3):Yes=0, No=3 $\rightarrow Gini=0$; Normal(1):Yes=1, No=3 $\rightarrow Gini=0$, 加权基尼 $\rightarrow 0$ ，所以基尼下降为 $0.375-0=0.375$ 。如果按照Temperature \Rightarrow Hot(2): Yes=0, No=2 $\rightarrow Gini=0$; Mild(1): Yes=0, No=1 $\rightarrow Gini=0$; Cool(1): Yes=1, No=0 $\rightarrow Gini=0$ ，所以基尼下降为 0.375 ，同理Windy的基尼下降为： 0.042 。最优：Humidity与Temperature并列最优（都能把Sunny子集分成纯叶，权重后 $Gini=0$ ）实现里通常按固定优先级/出现顺序/信息增益次要准则打破并列。这里任选其一即可，并且基尼指数降为0，所以自动停止划分。
- 子集：Weather=Rainy($Gini=0.375$, sample=4)，同理划分，可以得到如果按照Windy特征进行划分，基尼下降为 0.375 ，下降指数最大，所以选择该特征进行划分，最终选择该特征。且节点里的样本全是同一类（基尼指数为0）自动停止划分。
- 决策树：Weather?
 - Overcast: Yes
 - Sunny:
 - Humidity = High : No
 - Humidity = Normal: Yes
 - Rainy:
 - Windy = False: Yes
 - Windy = True : No

➤ 小结：1) CART用基尼指数作为划分准则，选择能最大降低基尼不纯度的特征；2) 递归构建二叉树，直到叶子节点纯或触发停止条件（如最大深度、最小样本数）。

▶ 模型求解：算法对比

➤ 各种算法有各自的特点，需要根据不同的应用场景进行选择：

算法	划分准则	树类型	适用任务	优点	缺点
ID3	信息增益	多叉树	分类	简单直观，首次引入熵	偏向取值多的属性，不能处理连续特征
C4.5	信息增益率	多叉树	分类	解决 ID3 偏好问题，可处理连续特征	计算复杂度较高，只能分类
CART	分类：基尼指数 回归：方差/MSE	二叉树	分类、回归	高效、统一结构、支持剪枝，是集成树的基石	只能二叉划分，容易过拟合，对噪声敏感

- ID3：ID3 用信息增益做划分，生成多叉分类树，简单直观，但偏好多值特征，容易过拟合。
- C4.5：C4.5 用信息增益率做划分，生成多叉分类树，能处理连续特征和缺失值，克服了 ID3 的偏好问题，但计算更复杂，仍可能过拟合。
- CART：CART 用“基尼指数/方差”做划分，生成二叉树，能做分类和回归，支持剪枝，是现代树模型的核心；但容易过拟合，对噪声敏感。

ID3 用信息增益划分生成多叉分类树，简单但易过拟合；C4.5 改进为信息增益率并支持连续特征与缺失值，更稳健但计算更复杂；CART 用基尼指数或方差生成二叉树，能做分类和回归，是现代树模型的核心。

▶ 模型评估

- 训练好决策树模型后，我们需要知道它在**未见过的数据**上的表现，即是否能很好地泛化，而不是仅仅记住训练数据。由于决策树能处理分类模型，所以可以用**分类性能指标**和**概率质量指标**进行评估：

类别	指标名称	公式	说明
分类性能指标	准确率	$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$	正确预测样本数 / 总样本数，整体正确性。
	精确率	$Precision = \frac{TP}{TP + FP}$	在预测为正类的样本中，真正为正类的比例。
	召回率/灵敏度	$Recall = \frac{TP}{TP + FN}$	在实际正类样本中，被正确预测为正类的比例。
	F1分数	$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$	精确率与召回率的调和平均，综合考量模型性能。
概率质量指标	ROC-AUC	$ROC : t \mapsto (FPR(t), TPR(t)), \quad t \in [0, 1]$	反映模型区分正负样本的能力，曲线下的面积越大越好。
	对数似然 (Log-Loss)	$\text{LogLoss} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$	度量预测概率与真实标签的差异，值越小说明概率预测越准确。
	Brier Score	$Brier Score = \frac{1}{N} \sum_{i=1}^N (p_i - y_i)^2$	预测概率与真实标签之间的均方误差，越小越好。

▶ 模型评估

- 训练好决策树模型后，我们需要知道它在**未见过的数据**上的表现，即是否能很好地泛化，而不是仅仅记住训练数据。由于决策树CART算法还能处理回归模型，所以还可以**回归性能指标**进行评估：

指标名称	缩写	公式	含义说明
均方误差	MSE	$MSE = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$	平均预测误差的平方，衡量整体误差
均方根误差	RMSE	$RMSE = \sqrt{MSE}$	MSE 的平方根，单位与原数据一致，更直观
决定系数	R ²	$R^2 = 1 - \frac{\sum(\hat{y}_i - y_i)^2}{\sum(\hat{y}_i - \bar{y})^2}$	表示模型对结果方差的解释能力，越接近1越好

决策树的评估在分类中看准确率、精确率、召回率、F1 与概率指标（对数损失、Brier 分数），在回归中看 MSE、MAE、R²与预测分布的方差拟合优度。

▶ 小结

- 决策树模型的求解主要依赖于递归划分与启发式搜索，核心算法包括 ID3、C4.5 与 CART，其中 CART 以基尼指数/方差为准则构建二叉树，并结合剪枝技术在分类与回归任务中实现了高效且实用的应用。
 - ID3：以信息增益为划分准则构建多叉树，能快速生成结构清晰的分类模型，但容易偏向取值多的特征。
 - C4.5：在 ID3 基础上改进为信息增益率，支持连续特征与缺失值处理，提升了稳定性与适用性。
 - **CART**：进一步发展为基于基尼指数或方差的二叉树算法，既能做分类又能做回归，并通过剪枝缓解过拟合，成为现代树模型的核心。
- 决策树模型的评估主要通过分类性能指标（准确率、召回率、F1 等）、概率质量指标（对数损失、Brier 分数等）、回归性能指标（MSE、RMSE、R²等）进行评估。

决策树的核心在于“构”与“简”——通过递归划分特征空间（基于信息增益、增益率或基尼指数）构建树模型，再通过分类或性能回归指标、概率质量指标进行评估。

目录章节

CONTENTS

01

决策树的概念和原理

02

模型求解与评估

03

模型扩展与改进

04

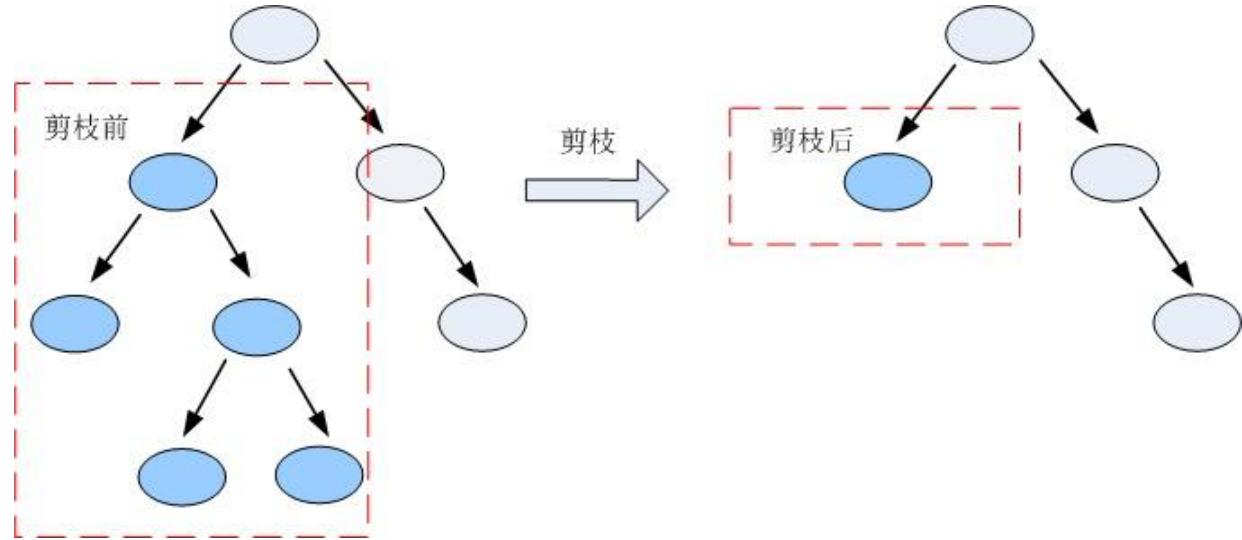
模型实现与实战

05

总结

► 模型扩展与改进——剪枝

- 决策树模型如果完全按训练集分裂时容易生成过深或过复杂的树，会出现问题：1. 树过深，分支太多 → 过拟合；2. 对噪声数据敏感 → 泛化能力下降；3. 模型复杂 → 可解释性差。
- 怎么进行解决？——**剪枝**，剪枝的核心目标主要是：1) 降低模型复杂度；2) 提高泛化能力；3) 减少噪声对模型的影响。剪枝通过删除或限制某些分支，使树更简单、更稳健。
- 本质：去掉对泛化能力贡献不大的分支，降低树的复杂度，在保持预测能力的前提下让模型更稳健。



- 剪枝也分为预剪枝（Pre-pruning）和后剪枝（Post-pruning），主要的区别在于两者在生成决策树进行剪枝的时机不同。预剪枝是边构造边剪枝，后剪枝是构造完后整体修剪。

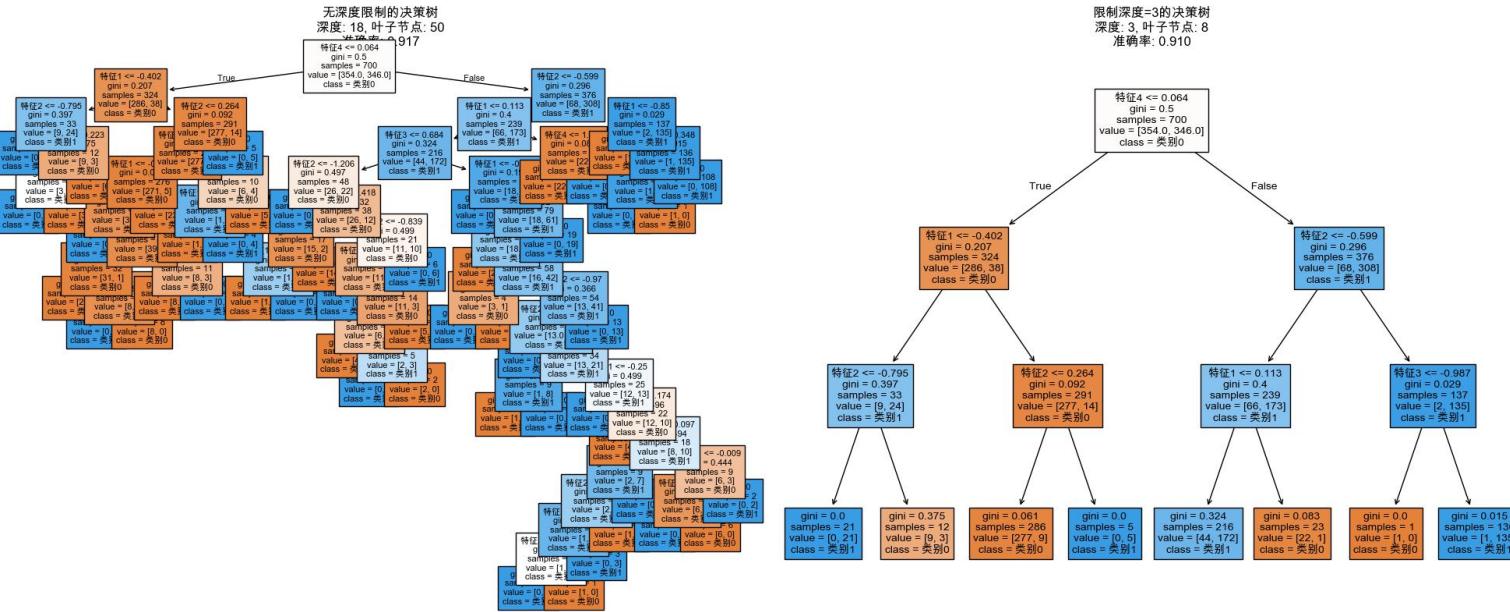
▶ 模型扩展与改进——剪枝

➤ 预剪枝 (Pre-pruning)

■ 定义：在生成决策树的过程中，对分裂节点进行约束，提前停止生长。

■ 常用条件：

● 1) 最大深度限制 (max_depth)：树的深度不能超过设定值，限制树的最大层数。



● 作用：防止树无限分裂导致过拟合。

● 最大深度过大 → 容易过拟合；最大深度过小 → 容易欠拟合。

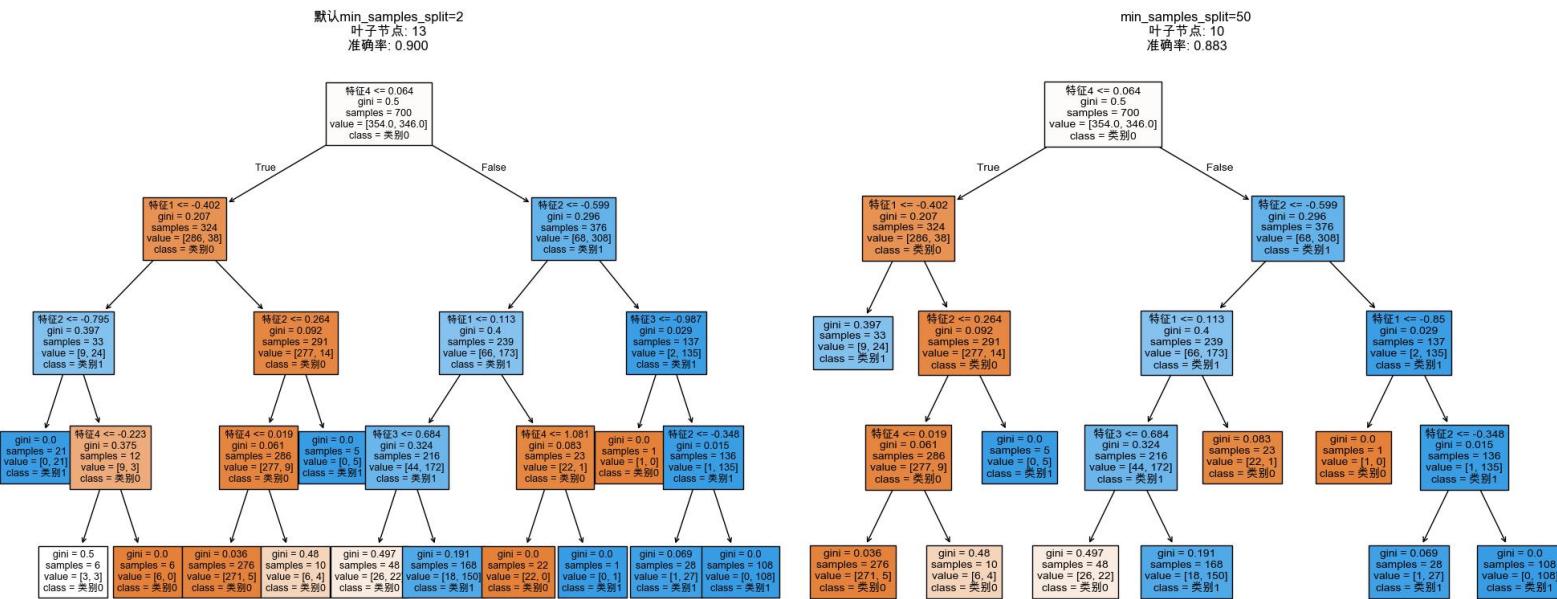
▶ 模型扩展与改进——剪枝

➤ 预剪枝 (Pre-pruning)

■ 定义：在生成决策树的过程中，对分裂节点进行约束，提前停止生长。

■ 常用条件：

- 2) **最小样本数限制 (min_samples_split)**：一个节点必须包含至少多少样本才能继续分裂。



- 作用：分裂后的样本数低于阈值，则不再划分，避免叶节点过少导致的不稳定。
- 最小样本树过小 → 叶子太碎，过拟合，模型不稳定；最小样本数过大 → 叶子太粗，欠拟合，模型简单。

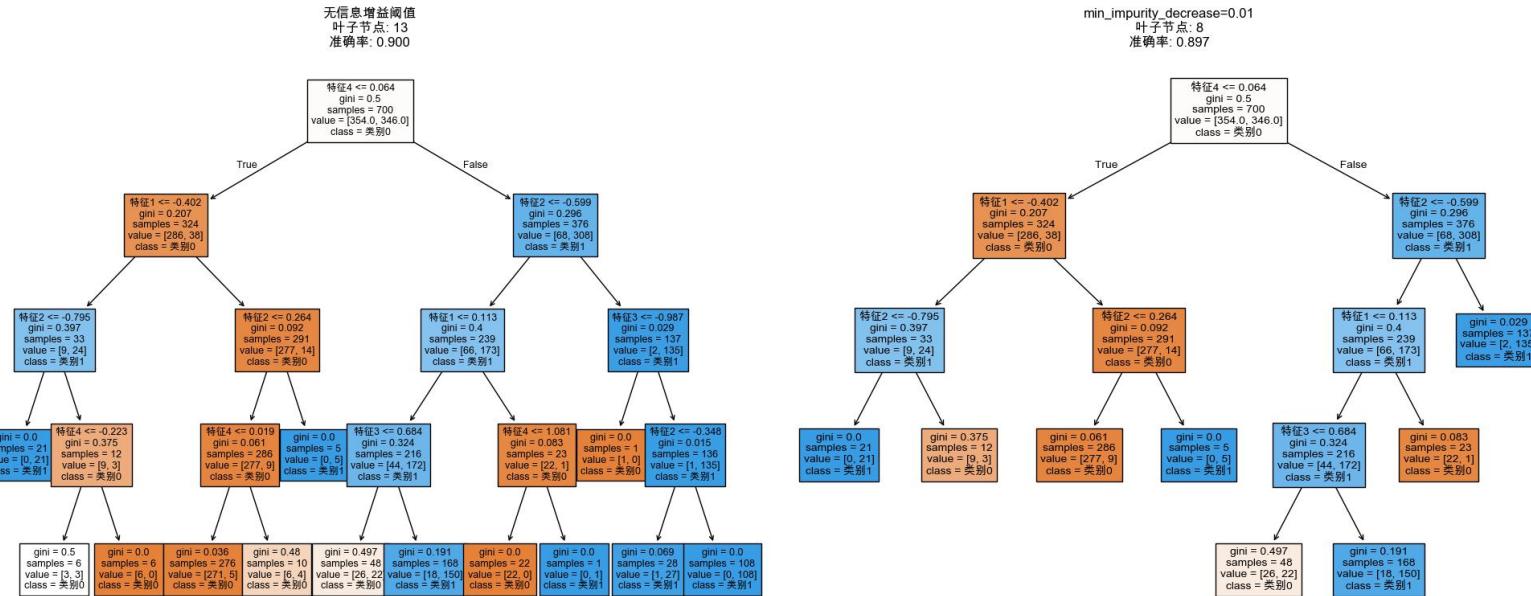
▶ 模型扩展与改进——剪枝

➤ 预剪枝 (Pre-pruning)

■ 定义：在生成决策树的过程中，对分裂节点进行约束，提前停止生长。

■ 常用条件：

- 3) 信息增益或基尼增益阈值 (`min_impurity_decrease / min_impurity_split`)：如果划分后的增益太小（低于 ϵ ），停止分裂，防止无意义的划分。



- 作用：避免划分带来的提升太小，导致无意义的分裂。
- 阈值过小 → 几乎所有划分都能通过 → 树复杂 → 过拟合；阈值过大 → 只有收益特别高的划分才会执行 → 树简单 → 欠拟合。

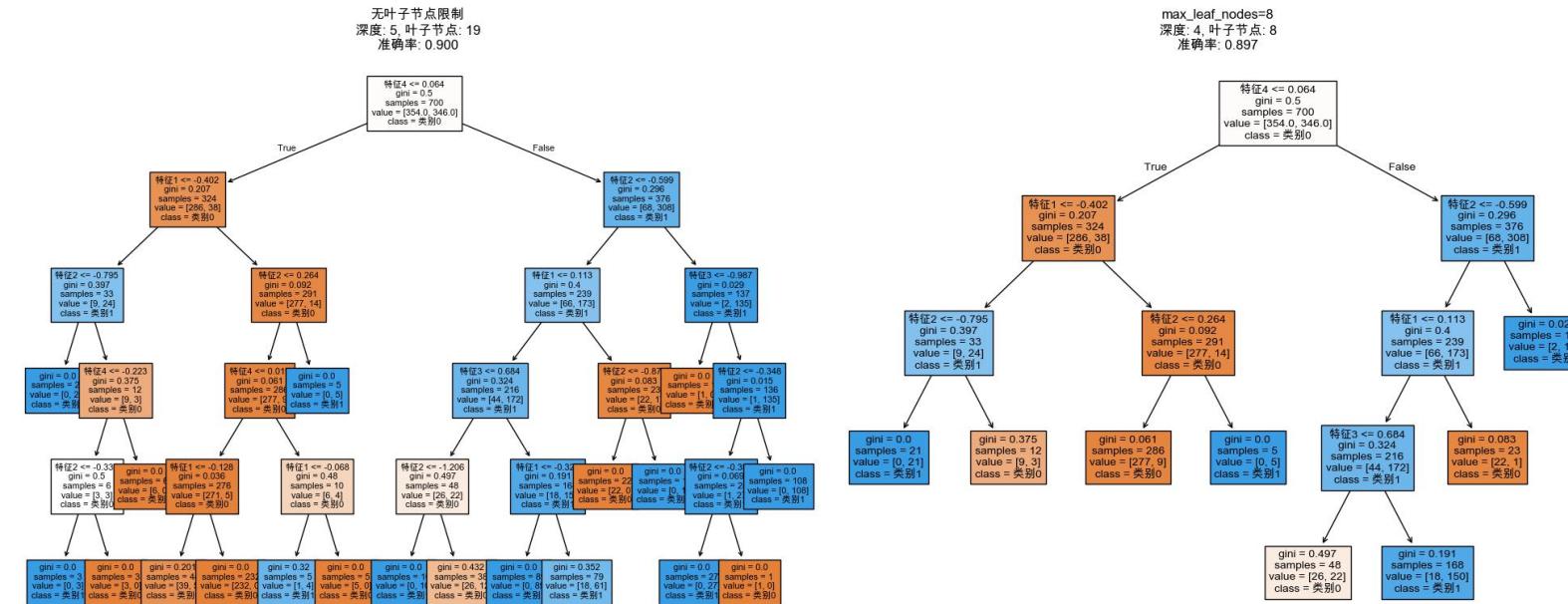
▶ 模型扩展与改进——剪枝

➤ 预剪枝 (Pre-pruning)

■ 定义：在生成决策树的过程中，对分裂节点进行约束，提前停止生长。

■ 常用条件：

- 4) 最大叶子节点数 (max_leaf_nodes) : 限制整棵树最多能有多少个叶子。



● 作用：间接控制树的复杂度。

- 最大叶子节点数过小 → 叶子太少，预测过于粗糙 → 欠拟合；最大叶子节点数过大 → 叶子过多，树复杂 → 过拟合。

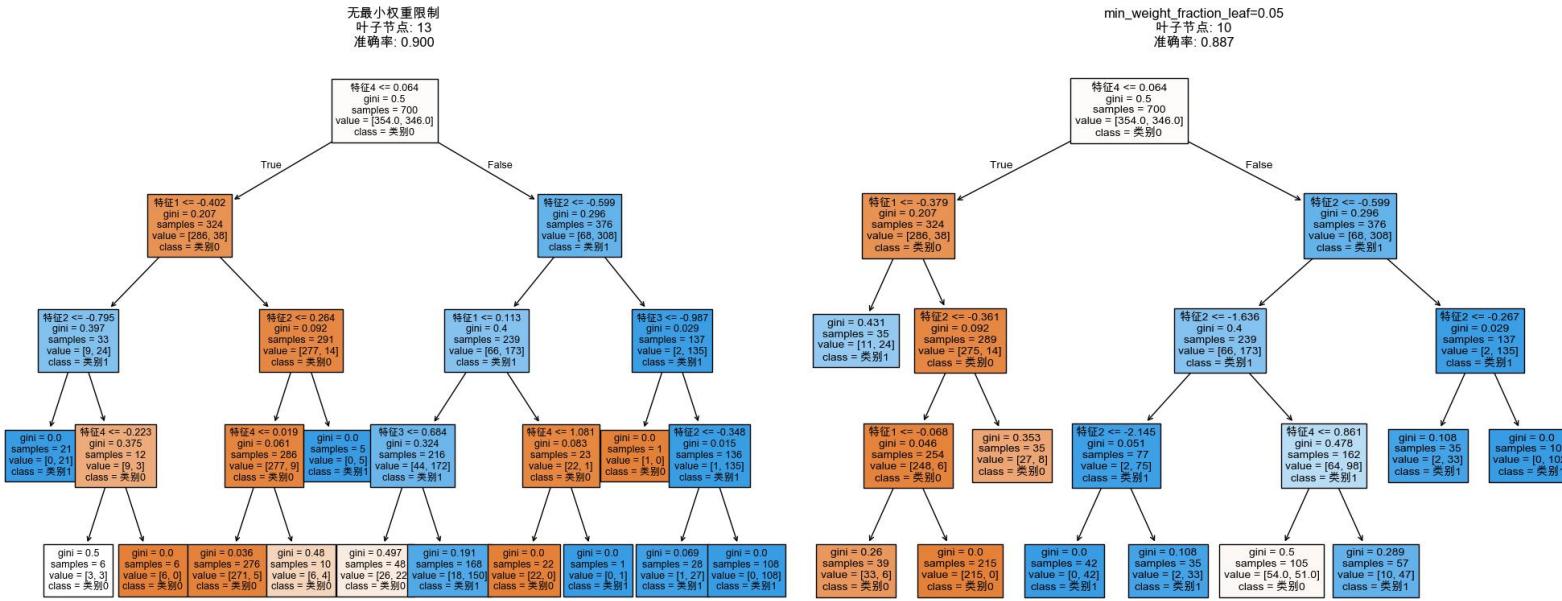
▶ 模型扩展与改进——剪枝

➤ 预剪枝 (Pre-pruning)

■ 定义：在生成决策树的过程中，对分裂节点进行约束，提前停止生长。

■ 常用条件：

- 5) **最小样本权重 (min_weight_fraction_leaf)**：要求每个叶子节点至少包含一定比例的总样本权重。



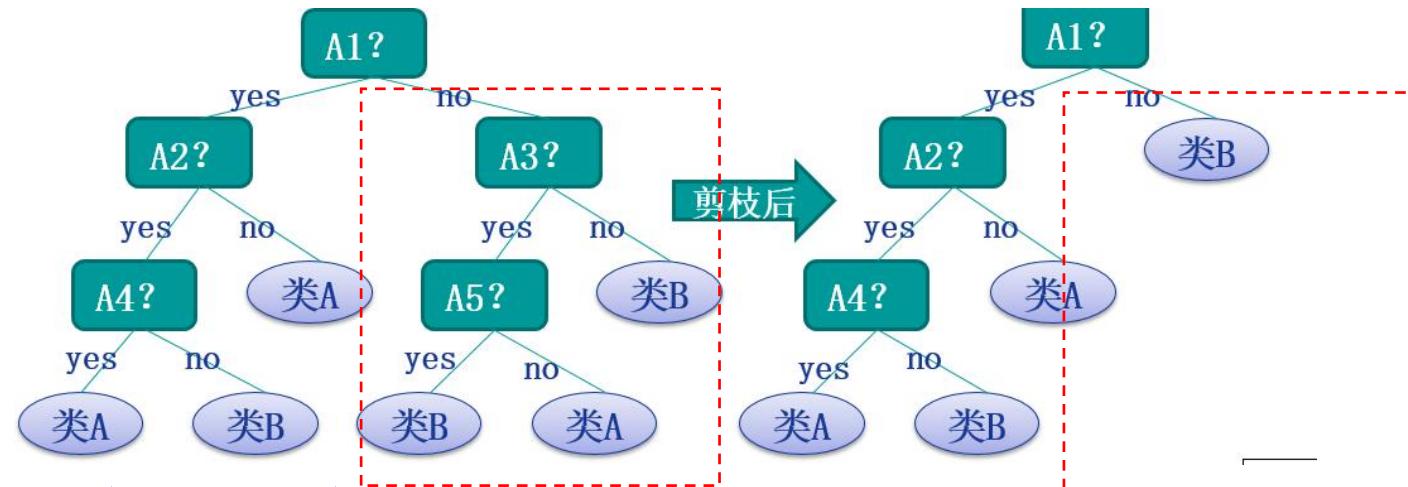
- 作用：在样本不均衡时防止过小叶子。

- 最小样本权重过小 → 少数样本也能形成叶子，树复杂 → 过拟合；
最小样本权重过大 → 许多小样本被忽略，树简单 → 欠拟合。

► 模型扩展与改进——剪枝

➤ 后剪枝 (Post-pruning)

- 定义：先让树完全生长，然后再评估每个节点是否需要剪掉子树，保留对泛化有用的分支。
- 1) 基于验证集剪枝：



- a. 将数据分为训练集和验证集。
- b. 利用训练集数据先生成一棵完全生长的树。
- c. 算法核心：对每个非叶节点：将节点转换为叶节点（删除其所有的子树），然后计算验证集误差。
- d. 如果误差下降或变化不大则保留剪枝，否则恢复原子树。

► 模型扩展与改进——剪枝

➤ 后剪枝 (Post-pruning)

- 定义：先让树完全生长，然后再评估每个节点是否需要剪掉子树，保留对泛化有用的分支。
 - 2) 成本复杂度剪枝 (Cost-Complexity Pruning, α -剪枝)：
 - 核心思想：在保持预测能力的前提下，引入**树复杂度惩罚项**，形成目标函数：
- $$R_\alpha(T) = R(T) + \alpha \cdot |T|$$
- 其中： $R(T)$ ：树 T 的误差（分类误差或回归误差）； $|T|$ ：叶节点数（树复杂度，通常用叶子节点个数来表示）； α ：控制复杂度惩罚的超参数 (>0)。
- a.**算法核心：对每个非叶节点：暂时将其子树剪掉，节点变为叶节点，然后计算 $R_\alpha(T)$ 。**
 - b.判断规则：如果剪掉后 $R_\alpha(T)$ 减少则保留剪枝，否则保留原子树。
 - c.重复直到无法进一步降低 $R_\alpha(T)$ 。
- 含义：如果只看 $R(T)$ ，如果树越复杂越大 (T 越大)，导师训练误差就越小，容易过拟合导致 $R(T)$ 越小，但是如果引入树复杂度惩罚项 ($\alpha|T|$)，之后，树越大就会被“惩罚”更多。所以**两者控制了“拟合误差”和“模型复杂度”之间的权衡**。

► 模型扩展与改进——集成方法

➤ 1) Bagging (Bootstrap Aggregating)

- 典型代表：随机森林（Random Forest）：每棵树在节点分裂时只考虑随机选择的特征子集（减少相关性，提高泛化能力）。通常使用**完全生长的树，不剪枝**。
- 原理：对训练集进行有放回采样（Bootstrap），生成多个子训练集，然后对每个子训练集训练一棵决策树。
- 优点：降低方差、减少过拟合。
- 缺点：生成大量树，内存消耗高；可解释性下降。

➤ 2) Boosting

- 典型代表：AdaBoost：对错误样本加权，提高后续树关注度；GBDT（Gradient Boosting Decision Tree）：将Boosting与梯度下降结合，每颗树拟合前一轮残差；XGBoost/LightGBM/CatBoost：对GBDT进行了系统优化：支持正则化、缺失值处理、并行/直方图加速。

- 原理：每棵树依赖前一棵树的预测误差，重点关注被前一棵树分类错误的样本：

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

$F_m(x)$: 第m棵树的累计预测，
 $h_m(x)$: 第m棵树的预测， γ : 学习率或步长。

- 优点：降低偏差，提高准确率；对数据噪声相对鲁棒（特别是 XGBoost、LightGBM）。
- 缺点：训练顺序依赖性强，难以并行化（LightGBM 通过优化部分解决）；对异常值敏感，需要调节学习率与树深度

▶ 模型扩展与改进——集成方法

➤ 3) 集成方法的核心技巧

- 行抽样 (Sample Subsampling)：减少过拟合，常用于随机森林和部分 Boosting。
- 列抽样 (Feature Subsampling)：降低特征相关性，提高鲁棒性。
- 树深度与叶子节点控制：Bagging 通常用深树，强调低偏差；Boosting 通常用浅树，减少过拟合风险。
- 学习率调节：Boosting 类算法使用学习率 γ 控制每棵树的贡献。
- 正则化：对叶子分数、树的复杂度进行约束，避免Boosting过拟合。

➤ 4) 适用场景总结：

方法	优点	典型应用
Bagging/Random Forest	稳健，减少过拟合，高维稀疏数据	信用评分、医学诊断、特征选择
Boosting/GBDT/XG Boost/LightGBM	高精度，适合复杂非线性问题	点击率预测、推荐系统、回归任务

集成方法是决策树最重要的扩展之一，通过组合多棵树（Bagging 降低方差，Boosting 降低偏差），显著提高模型稳定性和预测能力，并能适应分类、回归等多种任务，同时通过随机特征选择、样本抽样和正则化进一步控制过拟合。

► 模型扩展与改进——其他

➤ 1) 特征选择与划分准则优化

- 替换信息增益/增益率/基尼指数等，用更适合数据的准则（如多变量切分、最小方差或正则化增益）。
- 对连续特征可以尝试更精细的切分策略（不只是二分或单一阈值）。

➤ 2) 特征组合与高阶交互

- 构建组合特征（如特征交叉）或使用多变量划分（oblique split）提升模型表达能力。

➤ 3) 处理缺失值和异常值

- 决策树天然支持部分缺失值，可改进为对缺失特征选择最优分裂或使用 surrogate split。

➤ 4) 概率估计与置信度输出

- 不只输出类别或预测值，还估计概率分布（叶节点统计或贝叶斯平滑），便于概率性决策。

➤ 5) 多输出与多任务扩展

- 决策树可用于多输出回归、多标签分类等任务，通过共享树结构或叶节点输出向量实现。

▶ 小结

- 决策树的扩展与改进主要围绕**降低过拟合、提升泛化能力、支持更复杂任务、提高计算效率**几个方向，同时结合概率输出和特征交互增强模型表达力。

改进方向	方法	主要作用/场景	典型代表/应用
剪枝	预剪枝（限制深度、最小样本数）、后剪枝（验证集误差）	降低过拟合、提升泛化性	CART剪枝、C4.5剪枝
集成方法	Bagging、随机森林、Boosting (XGBoost、LightGBM、CatBoost)	提升精度与鲁棒性，减少方差/偏差	随机森林、梯度提升树
特征选择与划分优化	增益率、基尼指数、方差最小化、多变量切分 (Oblique split)	更合理的划分，适应不同数据特征	ID3, C4.5, CART
多任务扩展	多输出回归、多标签分类（叶节点输出向量）	支持更复杂的预测任务	多输出决策树
概率估计	叶节点统计 + 平滑处理（如拉普拉斯修正、贝叶斯平滑）	提供预测置信度，适用于概率性决策	概率型分类树
特征组合与交互	构建交叉特征、使用 oblique split（非轴对齐划分）	捕捉高阶特征关系，增强表达力	高阶交互树
处理缺失值和异常值	surrogate split 替代划分、加权处理缺失值、鲁棒分裂	增强对不完整/噪声数据的适应性	CART 缺失值处理

目录章节

CONTENTS

01

决策树的概念和原理

02

模型求解与评估

03

模型扩展与改进

04

模型实现与实战

05

总结

▶ 算法实现：numpy实现

➤ 实现：MyDecisionTree类，第一步：先定义类及初始化方法及内部（私有_）方法。

```
import numpy as np

class MyDecisionTree:
    def __init__(self, criterion="ID3", max_depth=None, min_samples_split=2,
                 min_impurity_decrease=0.0, max_leaf_nodes=None, min_weight_fraction_leaf=0.0):
        """
        criterion: "ID3", "C4.5", "CART"
        max_depth: 最大深度
        min_samples_split: 节点继续分裂所需的最小样本数
        min_impurity_decrease: 最小信息增益/基尼指数下降，低于该值不分裂
        max_leaf_nodes: 最大叶子节点数
        min_weight_fraction_leaf: 叶子节点至少包含的样本权重比例
        """
        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_impurity_decrease = min_impurity_decrease
        self.max_leaf_nodes = max_leaf_nodes
        self.min_weight_fraction_leaf = min_weight_fraction_leaf
        self.tree = None
        self._leaf_count = 0 # 跟踪当前叶子数量

    def _entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        probs = counts / counts.sum()
        return -np.sum(probs * np.log2(probs + 1e-9))

    def _gini(self, y):
        _, counts = np.unique(y, return_counts=True)
        probs = counts / counts.sum()
        return 1 - np.sum(probs ** 2)

    def _information_gain(self, X_column, y):
        H = self._entropy(y)
        values, counts = np.unique(X_column, return_counts=True)
        weighted_entropy = 0
        for v, c in zip(values, counts):
            weighted_entropy += (c/len(y)) * self._entropy(y[X_column == v])
        return H - weighted_entropy

    def _gain_ratio(self, X_column, y):
        info_gain = self._information_gain(X_column, y)
        _, counts = np.unique(X_column, return_counts=True)
        split_info = -np.sum((counts/len(y)) * np.log2(counts/len(y) + 1e-9))
        return info_gain / (split_info + 1e-9)

    def _gini_gain(self, X_column, y):
        G = self._gini(y)
        values, counts = np.unique(X_column, return_counts=True)
        weighted_gini = 0
        for v, c in zip(values, counts):
            weighted_gini += (c/len(y)) * self._gini(y[X_column == v])
        return G - weighted_gini

    def _calc_impurity_gain(self, X_column, y):
        if self.criterion == "ID3":
            return self._information_gain(X_column, y)
        elif self.criterion == "C4.5":
            return self._gain_ratio(X_column, y)
        elif self.criterion == "CART":
            return self._gini_gain(X_column, y)
        else:
            raise ValueError("Unsupported criterion")
```

- **类定义及初始化：**可选参数包括算法类型、最大深度、最小样本数、最小增益阈值、最大叶子节点数、叶子节点样本权重比例。
- **_entropy方法（_表示私有）：**计算熵。
- **_gini方法（_表示私有）：**计算基尼指数。
- **_information_gain方法（_表示私有）：**计算信息增益：
- **_gain_ratio方法（_表示私有）：**计算信息增益率：
- **_gini_gain方法（_表示私有）：**计算基尼指数增益：
- **_calc_impurity_gain方法（_表示私有）：**接口方法用于根据参数选择不同的算法计算增益。

▶ 算法实现：numpy实现

- 实现：MyDecisionTree类，第二步：实现核心的模型训练和预测方法：fit(X, y), best_split(X, y), build_tree(X, y), predict(X), predict_one(x, tree)。

```
def best_split(self, X, y):  
    n_features = X.shape[1]  
    best_feature, best_score = None, -1  
  
    for feature in range(n_features):  
        X_column = X[:, feature]  
        score = self._calc_impurity_gain(X_column, y)  
  
        if score > best_score:  
            best_score = score  
            best_feature = feature  
  
    return best_feature, best_score  
  
def build_tree(self, X, y, depth=0, total_samples=None):  
    if total_samples is None:  
        total_samples = len(y)  
  
    labels, counts = np.unique(y, return_counts=True)  
  
    if len(labels) == 1: # 纯节点  
        self._leaf_count += 1  
        return labels[0]  
    if self.max_depth is not None and depth >= self.max_depth:  
        self._leaf_count += 1  
        return labels[np.argmax(counts)]  
    if len(y) < self.min_samples_split:  
        self._leaf_count += 1  
        return labels[np.argmax(counts)]  
    if self.max_leaf_nodes is not None and self._leaf_count >= self.max_leaf_nodes:  
        return labels[np.argmax(counts)]  
  
    best_feature, best_score = self.best_split(X, y)  
    if best_feature is None or best_score < self.min_impurity_decrease:  
        self._leaf_count += 1  
        return labels[np.argmax(counts)]  
  
    tree = {"feature": best_feature, "children": {}}  
    values = np.unique(X[:, best_feature])  
  
    for v in values:  
        idx = X[:, best_feature] == v  
        # min_weight_fraction_leaf 检查  
        if sum(idx) < self.min_weight_fraction_leaf * total_samples:  
            # 如果不满足要求 → 直接当作叶子  
            tree["children"][v] = labels[np.argmax(counts)]  
        else:  
            subtree = self.build_tree(X[idx], y[idx], depth+1, total_samples)  
            tree["children"][v] = subtree  
  
    return tree  
  
def fit(self, X, y):  
    self._leaf_count = 0  
    self.tree = self.build_tree(X, y)
```



```
def predict_one(self, x, tree):  
    if not isinstance(tree, dict):  
        return tree  
    feature = tree["feature"]  
    value = x[feature]  
    if value in tree["children"]:  
        return self.predict_one(x, tree["children"][value])  
    else:  
        return None  
  
def predict(self, X):  
    return np.array([self.predict_one(x, self.tree) for x in X])
```

- 决策树训练流程：1) 特征选择：计算各特征的信息增益/基尼指数 → 选出最优划分特征；2) 数据划分：根据最优特征的取值将样本分配到子节点；3) 递归建树：对子节点重复执行特征选择与划分；4) 停止条件：当节点纯度足够高、样本过少或达到最大深度/叶子数时，生成叶节点。

- 决策树预测流程：遍历测试样本 → 从根节点开始根据当前特征的取值进入对应子节点 → 递归向下直至到达叶子节点 → 返回该叶节点的类别作为预测结果 → 汇总所有样本的预测标签。

► 算法实现：numpy实现

- 实现：MyDecisionTree类，第三步：实现评估方法。

```
def score(self, X, y):  
    """ 计算准确率 """  
    y_pred = self.predict(X)  
    return np.mean(y_pred == y)
```

- 这里只实现了分类的作用，所以只实现分类的准确率指标。

▶ 算法实现：pytorch实现

- 利用Pytorch实现，两个版本几乎一致，只是将numpy包转换为pytorch包，不需要掌握了解即可。

```
import torch

class MyDecisionTree_Torch:
    def __init__(self, criterion="ID3", max_depth=None, min_samples_split=2,
                 min_impurity_decrease=0.0, max_leaf_nodes=None, min_weight_fraction_leaf=0.0):
        ...
        criterion: "ID3", "C4.5", "CART"
        max_depth: 最大深度
        min_samples_split: 节点继续分裂所需的最小样本数
        min_impurity_decrease: 最小信息增益/基尼指数下降，低于该值不分裂
        max_leaf_nodes: 最大叶子节点数
        min_weight_fraction_leaf: 叶子节点至少包含的样本权重比例
        ...
        self.criterion = criterion
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.min_impurity_decrease = min_impurity_decrease
        self.max_leaf_nodes = max_leaf_nodes
        self.min_weight_fraction_leaf = min_weight_fraction_leaf
        self.tree = None
        self._leaf_count = 0 # 跟踪当前叶子数量

    def _entropy(self, y):
        _, counts = torch.unique(y, return_counts=True)
        probs = counts.float() / counts.sum()
        return -(probs * torch.log2(probs + 1e-9)).sum()

    def _gini(self, y):
        _, counts = torch.unique(y, return_counts=True)
        probs = counts.float() / counts.sum()
        return 1 - (probs ** 2).sum()

    def _information_gain(self, X_column, y):
        H = self._entropy(y)
        values, counts = torch.unique(X_column, return_counts=True)
        weighted_entropy = torch.tensor(0.0)
        for v, c in zip(values, counts):
            weighted_entropy += (c.float() / len(y)) * self._entropy(y[X_column == v])
        return H - weighted_entropy

    def _gain_ratio(self, X_column, y):
        info_gain = self._information_gain(X_column, y)
        _, counts = torch.unique(X_column, return_counts=True)
        probs = counts.float() / len(y)
        split_info = -(probs * torch.log2(probs + 1e-9)).sum()
        return info_gain / (split_info + 1e-9)

    def _gini_gain(self, X_column, y):
        G = self._gini(y)
        values, counts = torch.unique(X_column, return_counts=True)
        weighted_gini = torch.tensor(0.0)
        for v, c in zip(values, counts):
            weighted_gini += (c.float() / len(y)) * self._gini(y[X_column == v])
        return G - weighted_gini

    def _calc_impurity_gain(self, X_column, y):
        if self.criterion == "ID3":
            return self._information_gain(X_column, y)
        elif self.criterion == "C4.5":
            return self._gain_ratio(X_column, y)
        elif self.criterion == "CART":
            return self._gini_gain(X_column, y)
        else:
            raise ValueError("Unsupported criterion")
```

```
def best_split(self, X, y):
    n_features = X.shape[1]
    best_feature, best_score = None, -1

    for feature in range(n_features):
        X_column = X[:, feature]
        score = self._calc_impurity_gain(X_column, y)

        if score > best_score:
            best_score = score
            best_feature = feature

    return best_feature, best_score

def build_tree(self, X, y, depth=0, total_samples=None):
    if total_samples is None:
        total_samples = len(y)

    labels, counts = torch.unique(y, return_counts=True)

    if len(labels) == 1: # 纯节点
        self._leaf_count += 1
        return labels[0].item()
    if self.max_depth is not None and depth >= self.max_depth:
        self._leaf_count += 1
        return labels[counts.argmax()].item()
    if len(y) < self.min_samples_split:
        self._leaf_count += 1
        return labels[counts.argmax()].item()
    if self.max_leaf_nodes is not None and self._leaf_count >= self.max_leaf_nodes:
        self._leaf_count += 1
        return labels[counts.argmax()].item()

    best_feature, best_score = self.best_split(X, y)
    if best_feature is None or best_score < self.min_impurity_decrease:
        self._leaf_count += 1
        return labels[counts.argmax()].item()

    tree = {"feature": best_feature, "children": {}}
    values = torch.unique(X[:, best_feature])

    for v in values:
        idx = X[:, best_feature] == v
        if idx.sum() < self.min_weight_fraction_leaf * total_samples:
            tree["children"][v.item()] = labels[counts.argmax()].item()
        else:
            subtree = self.build_tree(X[idx], y[idx], depth+1, total_samples)
            tree["children"][v.item()] = subtree

    return tree

def fit(self, X, y):
    self._leaf_count = 0
    self.tree = self.build_tree(X, y)

def predict_one(self, x, tree):
    if not isinstance(tree, dict):
        return tree
    feature = tree["feature"]
    value = x[feature].item()
    if value in tree["children"]:
        return self.predict_one(x, tree["children"][value])
    else:
        return None

def predict(self, X):
    return torch.tensor([self.predict_one(x, self.tree) for x in X])

def score(self, X, y):
    y_pred = self.predict(X)
    return (y_pred == y).float().mean().item()
```

▶ 算法实战：客户流失预测

➤ 如之前一样，首先第一步就是导入数据，并查看数据的情况，决定下一步该干什么。

```
import pandas as pd

# 读取数据
data = pd.read_csv('Churn_Modelling.csv')

# 显示数据基本信息
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   RowNumber        10000 non-null   int64  
 1   CustomerId      10000 non-null   int64  
 2   Surname          10000 non-null   object  
 3   CreditScore      10000 non-null   int64  
 4   Geography         10000 non-null   object  
 5   Gender            10000 non-null   object  
 6   Age               10000 non-null   int64  
 7   Tenure            10000 non-null   int64  
 8   Balance           10000 non-null   float64 
 9   NumOfProducts     10000 non-null   int64  
 10  HasCrCard        10000 non-null   int64  
 11  IsActiveMember    10000 non-null   int64  
 12  EstimatedSalary   10000 non-null   float64 
 13  Exited            10000 non-null   int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProd
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

- RowNumber, CustomerId, Surname都是Object数据类型，但是这些都是没有用的特征，可以直接drop。所有数据都是non-null，不需要进行后续的缺失值处理。
- 其中Gender, Geography是有用的类别特征，需要转换为数值，方法包括标签编码（转换为0/1）、Onehot编码（转换为多个二进制特征）。

▶ 算法实战：房价预测

➤ 根据上面的分析，进行数据预处理，并进行数据集划分（训练集、测试集）。

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
# 数据预处理
X = data.drop(["RowNumber", "CustomerId", "Surname", "Exited"], axis=1)
y = data["Exited"]

# 数据中有 Geography 和 Gender 是类别型，需要转换为数值。
# 方法1：标签编码（适合二分类，转换为0/1）
le_gender = LabelEncoder()
X[["Gender"]] = le_gender.fit_transform(X[["Gender"]])

# 方法2：独热编码（适合多分类，转换为多个二进制特征）
X = pd.get_dummies(X, columns=["Geography"], drop_first=True).astype(int)

# 查看预处理后的数据
print("前5行数据:", X.head())
print("所有特征:", X.columns)

# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

前5行数据:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
0	619	0	42	2	0	1	1	
1	608	0	41	1	83807	1	0	
2	502	0	42	8	159660	3	1	
3	699	0	39	1	0	2	0	
4	850	0	43	2	125510	1	1	

所有特征: Index(['CreditScore', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Geography_Germany', 'Geography_Spain'], dtype='object')

● 预处理之后，所有的特征都变成了数值量，可以进行下一步的模型构建、预测与评估。

▶ 算法实战：房价预测

- 定义模型进行训练、预测，最后通过准确率、召回率、F1分数等指标评估效果。

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
clf = DecisionTreeClassifier(
    criterion="gini", # 取值: {"gini", "entropy", "log_loss"}
    max_depth=5,
    min_samples_split=20,
    random_state=42
)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print("准确率:", accuracy_score(y_test, y_pred))
print("\n分类报告:\n", classification_report(y_test, y_pred))
print("\n混淆矩阵:\n", confusion_matrix(y_test, y_pred))
```

准确率: 0.856

分类报告:

	precision	recall	f1-score	support
0	0.86	0.97	0.91	1593
1	0.79	0.40	0.53	407
accuracy			0.86	2000
macro avg	0.83	0.69	0.72	2000
weighted avg	0.85	0.86	0.84	2000

混淆矩阵:

```
[[1549  44]
 [ 244 163]]
```

- 模型整体准确率不错（约 86%），但对流失客户（类别 1）的识别能力较弱，召回率只有 40%，说明存在较多漏报。

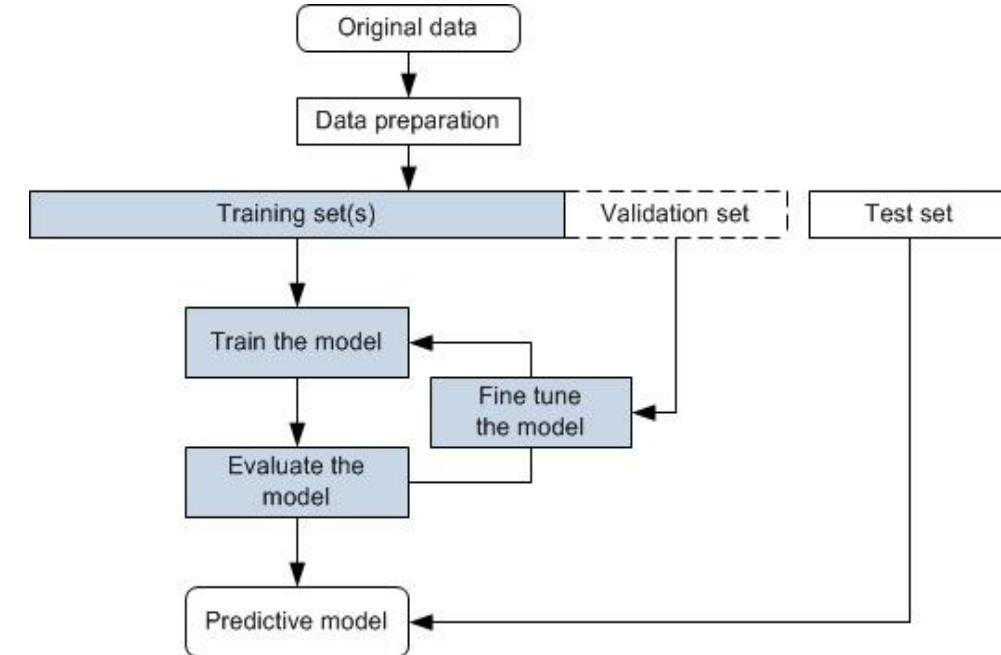
▶ 小结

➤ 模型实现:

- 主要的方法：初始化参数（`__init__`），拟合方法（`fit`），构建树方法（`build_tree`），最佳划分方法（`best_split`），预测方法（`predict`），评估指标（`score`），实现这些方法的私有方法（`_xxx`）。

➤ 模型实战：

- 数据准备：导入数据、查看数据。
- 数据划分：训练集 / 测试集（常用70%/30%或80%/20%）。
- 特征预处理：标准化/归一化/去NULL值/独热编码等。
- 决策树训练：递归按最优特征分裂节点，预测通过叶子节点输出类别。
- 模型评估：通过准确率，精准率，召回率，F1分数等进行评估。



决策树核心是训练时递归选择最优特征划分样本构建树，每个叶子节点对应类别；预测时遍历树节点，根据特征路径到叶子输出类别，可用准确率等评估模型性能。

目录章节

CONTENTS

01

决策树的概念和原理

02

模型求解与评估

03

模型扩展与改进

04

模型实现与实战

05

总结

▶ 总结

➤ 决策树的概念与原理

- ✓ 决策树是一种以树状结构对数据进行分类或回归的模型，每个节点根据特征划分数据以最大化信息增益或最小化不纯度。它通过从根节点到叶节点的路径，将输入映射到最终预测结果。
- ✓ 决策树直观、易解释，并且能够处理非线性关系和多类别特征，也是后续其他树模型的基础。
- ✓ 模型假设：1) 特征空间可被递归地划分成若干个区域；2) 每个区域内部是相对“纯净”的；3) 预测方式是分区常数模型。

➤ 决策树的求解与评估

- ✓ 决策树通过递归地选择最优特征划分节点（通常基于信息增益、信息增益率或基尼指数），逐步生成树结构；构建完成后，可通过准确率、混淆矩阵、ROC曲线等指标评估分类性能，或通过均方误差（MSE）、均方根误差（RMSE）、 R^2 等指标评估回归性能。

➤ 决策树的扩展与改进

- ✓ 决策树的扩展与改进主要有：1) 剪枝；2) 集成方法；3) 特征选择与划分优化；4) 多任务扩展；5) 概率估计；6) 特征组合与交互；7) 处理缺失值与异常值等。

决策树通过递归按最优特征划分样本构建树结构实现分类或回归预测，叶子节点输出最终结果，可用准确率或回归指标评估性能，同时可通过控制树深度、最小样本分裂数等参数防止过拟合并增强泛化能力。

感谢聆听



Personal Website: <https://www.miaopeng.info/>



Email: miaopeng@stu.scu.edu.cn



Github: <https://github.com/MMeowhite>



Youtube: <https://www.youtube.com/@pengmiao-bmm>