预剪枝与四种后剪枝算法

前言

上一节我们介绍了关于决策树的划分选择，我们介绍了关于信息增益（ID3）、增益率（C4.5）、基尼指数（CART），但是我们基于上述算法总结出来的算法都可能导致最终的模型出现过拟合的情况。所以我们需要考虑怎么减少模型“太过较真”。

什么是剪枝?

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，有时会造成决策树分支过多，这时就可能因训练样本学得“太好”了，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。

关于一些各位需要知道的基本模型原则：

- **对训练数据过拟合，**树的结构过于复杂，学习到了训练数据中不具普遍性的特殊规律。
- **泛化能力弱，**复杂模型对数据变化更为敏感，在测试数据上表现会显著下降。
- **奥卡姆剃刀原理（Occam's Razor），**在所有性能相当的模型中，我们应该选择最简单的那个。简单的模型更不容易过拟合。

因此，我们需要在决策树完全生长之后（或在其生长过程中）对其进行剪枝，以期获得一个**结构更简单、泛化能力更强**的子树。

剪枝：预剪枝和后剪枝

决策树剪枝的基本策略有“预剪枝”(prepruning)和“后剪枝”(postpruning) [Quinlan, 1993]。***预剪枝是指在决策树生成过程中，对每个结点在划分前先进行估计，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结点；后剪枝则是先从训练集生成一棵完整的决策树，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***。

预剪枝

预剪枝在决策树的**生成过程中**进行。在每次划分节点之前，先使用一个“停止条件”进行判断。如果当前节点的划分不能带来**泛化性能**的提升，就停止划分并将当前节点标记为叶节点。

那么这时候有人要问了。说主播主播，到底什么时候停止呢？这个条件是什么呢？好吧，这将会是一个值得谈论的问题，我们常见的停止划分的条件有：

1. **树达到最大深度（Max Depth）**：限制树的最大生长深度。

2. **节点样本数小于阈值（Min Samples Split）**：当前节点所含样本数少于某个预定值时，不再划分，以防止因样本过少而产生不具代表性的划分。
3. **叶节点样本数最小阈值（Min Samples Leaf）**：保证划分后每个子节点（叶节点）至少包含一定数量的样本。
4. **节点纯度/不纯度阈值**：当节点的基尼指数（Gini Index）或信息熵（Information Entropy）小于某个阈值时，说明样本已足够“纯”，无需再划分。
5. **性能提升不显著**：通过**留出法（Hold-out）**，计算划分前后在**验证集（Validation Set）**上的精度。如果划分不能显著提升验证集精度（例如，提升幅度小于某个阈值 ϵ），则停止划分。

在西瓜书中，则是使用第 5 种方法，这部分可以参考西瓜书P79-82。简单来说西瓜书当中的那个例子是基于验证集评估的预剪枝。***其核心是在每个节点划分前，用验证集精度作为判断标准，只有当划分能带来泛化性能的显著提升时，才进行划分，否则将该节点标记为叶结点***。

但是，*即便预剪枝显著减少训练时间和开销，也避免了许多不必要的划分。但是存在**欠拟合（Underfitting）**的风险。因为某些当前看起来“提升不显著”的划分，其后续的划分可能会极大地改善模型性能。而预剪枝本身是基于“贪心”（Greedy）的的，禁止这些分支展开，可能失去了学习更好模型的机会*。

来自西瓜书：一棵仅有一层划分的决策树，亦称“决策树桩”(decision stump)。

后剪枝

于是乎，我们就来到了后剪枝...***后剪枝是更常用、也更有效的策略。它先让决策树完全生长，得到一个可能过拟合的复杂树，然后自底向上地对非叶节点进行考察，判断是否将该节点替换为叶节点（即剪掉其下属的整个子树）。***

关于后剪枝，我想我有必要展开讲讲，因为在这方面有很多杰出的Paper提出的思想方法，这是我们需要作为学习的对象。这些方法包括：**REP、EBP/PEP、MEP、CCP**。我们一个一个来看（这部分西瓜书没有详细的提，所以我出手了）。

四种后剪枝算法

Reduced Error Pruning (REP, 简化错误剪枝)

关于REP，这个算法是西瓜书当中所提到的例子也就是：直接用**验证集**来评估子树是否能带来更好效果。好吧，很直觉的方法。你们可以参考西瓜书P82-83的解释...我还是要提一嘴，来说说这个算法。

这个算法的步骤在西瓜书里也有写，总结起来就是：***首先用训练集（通常占 2/3）生成一棵完整决策树，然后准备一个单独的验证集（通常占 1/3），对每个非叶子节点，尝试剪掉该子树并替换为叶子（预测该节点上样本最多的类别）。如果在验证集上的准确率没有下降（甚至提高），则执行剪枝。重复，直到无法进一步提升。***

需要注意的是，***使用验证集是一种策略，不要以为预剪枝也是REP，只不过他们都使用了验证集而已***（这是我的疑问，也是最后我得到的答案）

这里引用原论文，感兴趣可以一看：Quinlan, J. R. (1987). *Simplifying decision trees*. International Journal of Man-Machine Studies.

其决策逻辑可以如下表述：

对于一个待剪枝的节点 n

- 设 E_{before} 为验证集中被节点 n 下的整个子树错误分类的样本数量。
- 设 E_{after} 为验证集中被替换后的叶节点错误分类的样本数量。（叶节点的类别是 $C_{majority}$，即节点 n 的训练样本中最普遍的类别）

剪枝条件为：

$$E_{after} \leq E_{before}$$

只要这个条件满足，就进行剪枝。很简单对吧，但是又过于依赖验证集（数据划分问题），而且剪枝过于激进了有点。

Error-Based Pruning / Pessimistic Error Pruning (EBP / PEP, 悲观错误剪枝)

同样是剪枝EBP（这两者的区别：**PEP** 强调“悲观估计”这一点，而**EBP** 是更宽泛的说法，表示“用误差来做剪枝”）就***不需要验证集，直接在训练集上对错误率进行统计校正（乐观估计→悲观估计）***。

我们来看看它的执行过程：

1. 观察误差率 (Observed Error Rate)：

对于一个节点 t ，设 $N(t)$ 是到达该节点的训练样本总数，其中 $E(t)$ 个被错误分类。

其观察误差率为: $e(t) = \dfrac{E(t)}{N(t)}$

2. 悲观误差率 (Pessimistic Error Rate)：

我们不相信 $e(t)$ 是真实的误差率。PEP 假设真实误差率服从一个二项分布，并使用**正态分布上限估计**（类似置信区间的上界）来得到一个更保守、更“悲观”的误差估计。

Quinlan 提出了一种简化且实用的计算方法，引入了**连续性校正 (Continuity Correction)**：

$$e'(t) = \frac{E(t) + \frac{1}{2}}{N(t)}$$

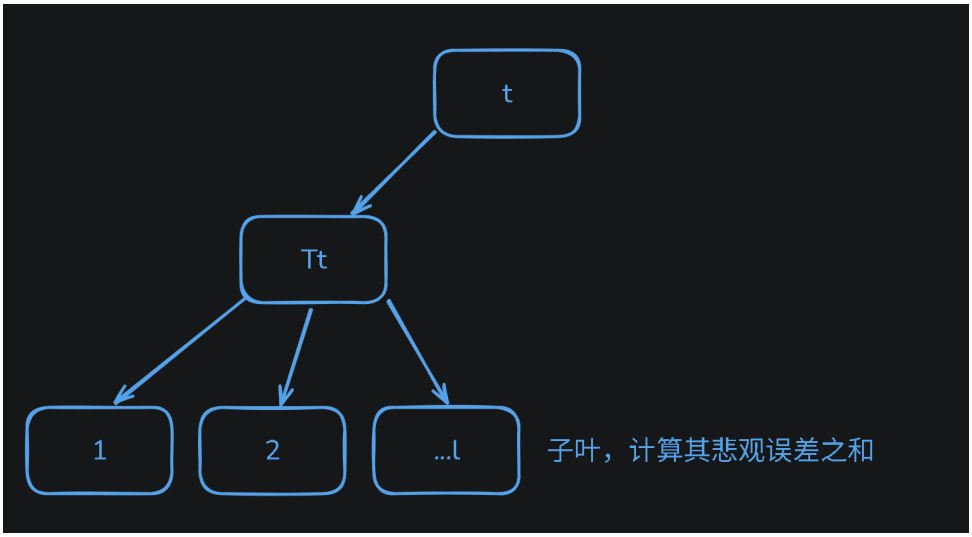
这里的 $\dfrac{1}{2}$ 就是连续性校正项。它相当于假设在最坏的情况下，我们可能多错了半个样本。这个校正使得估计更加保守。

3. 子树与叶节点的误差比较：

- 对于一个**子树** T_t （以节点 t 为根的子树），其悲观误差是它所有**叶节点** l 的悲观误差之和（因为每个样本最终都会到达一个叶节点并被分类）：

$$\text{Error}_{pessimistic}(T_t) = \sum_{l \in \text{leaves of } T_t} (E(l) + \frac{1}{2})$$

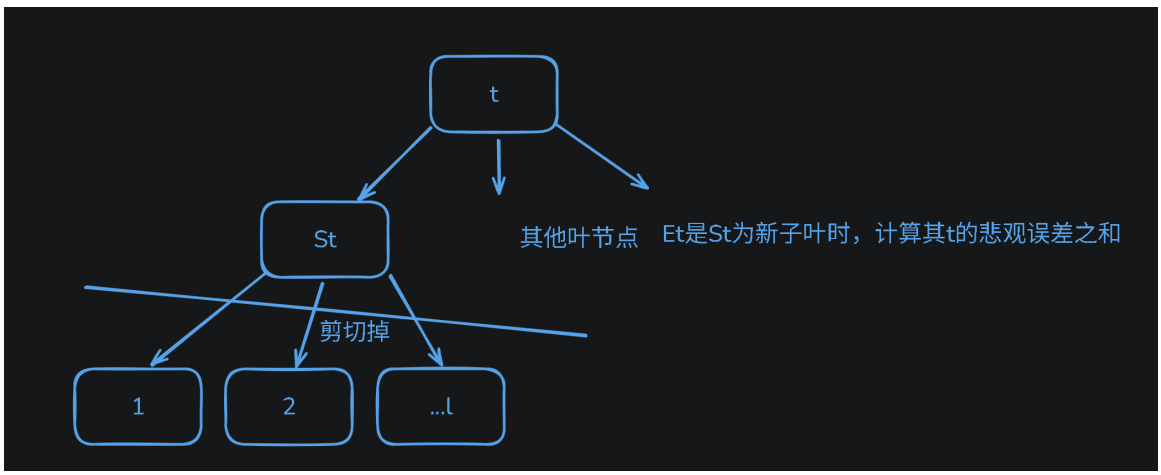
可以理解为，*整个子树的错误数估计是各叶节点错误数加二分之一后的总和*。如下图：



- 如果将其**剪枝为一个叶节点** S_t ，那么这个新叶节点的悲观误差为：

$$\text{Error}_{pessimistic}(S_t) = E(t) + \frac{1}{2}$$

注意：这里 $E(t)$ 是节点本身为新叶节点时，节点 t 的错误数。新叶节点的类别是节点 t 下的训练样本中最普遍的类别。$E(t)$ 就是这个新类别在节点 t 的样本上产生的错误数。如下图：



4. 剪枝决策规则：

如果 将子树替换为叶节点后的悲观误差 **小于或等于** 保留子树时的悲观误差，则进行剪枝。

$$\text{If } (E(t) + \frac{1}{2}) \leq \sum_{l \in \text{leaves of } T_t} (E(l) + \frac{1}{2})$$

那么 *剪枝。化简这个不等式，决策规则通常写为：*

$$\text{If } E(t) \leq \sum_l E(l) + \frac{(\text{number of leaves in } T_t) - 1}{2}$$

这个形式更直观：***如果节点 t 本身的错误数，小于其所有子叶节点的错误数之和再加上一个惩罚项（该子树每多一个叶节点，惩罚项就增加 0.5），那么就剪枝***。惩罚项体现了模型复杂度的代价。

同样的贴出原论文：Quinlan, J. R. (1987). *Simplifying decision trees.* IJMMS.

虽然这个算法不需要我们准备什么验证集，但是它*过于依赖统计估计，并且这个 $\frac{1}{2}$ 在实际使用的时候不够精准可能有点问题，导致模型出现误差*。

Minimum Error Pruning (MEP, 最小错误剪枝)

*同样也是基于**误差估计**，但 MEP 采用**更严格的统计检验**（通常基于置信区间），而不是使用 $\frac{1}{2}$。MEP 同样自底向上遍历。但不同的是，对于每个内部结点 t 比较的是两种情形的**期望错误率**（error rate）：*

- 把该结点**剪成叶**（用该结点的多数类做预测）时的**静态错误率（static error, STE）**，
- 保留该结点（即保持原子树）的**动态错误率（dynamic error, DYE）**

如果 $\mathrm{STE}(t) \leq \mathrm{DYE}(t)$ 就 **剪枝**。该方法使用一种带平滑/先验的概率估计（称为 m-probability 或 Laplace／m-estimate）来估计类别概率。(ResearchGate)

看看公式：

1. m-概率估计（m-probability）

设在结点 t 上共有 $N(t)$ 个训练样本，类别有 k 类（在 t 上有 k 类）；用 $n_i(t)$ 表示到达 t 的、属于第 i 类的样本数；先验（a-priori）第 i 类概率为 p_i（常取均匀 $p_i = 1/k$）。

对结点 t 中第 i 类的后验估计：

$$\hat{p}_i(t) = \frac{n_i(t) + m \cdot p_i}{N(t) + m}$$

其中 m 为平滑参数（m 越大越偏向先验）。

关于这个公式，你可以看看AI的回答：

- **贝叶斯估计**：
  给类别概率一个先验分布（通常是 **Dirichlet 分布**），再结合观测数据，得到后验分布的期望。

  如果先验是 Dirichlet($\alpha_1, \ldots, \alpha_k$)，那么后验的期望是

  $$\hat{p}_i^{Bayes} = \frac{n_i + \alpha_i}{N + \sum_j \alpha_j}$$

这里的 $\alpha_i = m \cdot p_i$，于是就有了***m-估计***

2. 结点的期望错误率（对单个到达该结点的新样本）

若用多数类（即使 $\hat{p}_{max}(t) = \max_i \hat{p}_i(t)$ ）来预测，结点上的**期望错误率（比例）**为

$$\mathrm{EER}(t) \;=\; 1 - \hat{p}_{max}(t)$$

注意这里是“率”（0 到 1），不是绝对错误个数。

3. 静态错误率（static error，剪成叶后）

对结点 t ，若把它直接替为叶节点（用该结点的多数类预测），则：

$$\mathrm{STE}(t) \;=\; \mathrm{EER}(t) \;=\; 1 - \hat{p}_{max}(t).$$

4. 动态错误率（dynamic error，剪子树前时）

观测落到 t 的样本按概率会流到各子结点 $c \in \mathrm{children}(t)$ 。设第 j 个子结点到达概率约为 $\dfrac{N(c_j)}{N(t)}$ （用训练样本比例作近似），子结点的期望错误率为 $\mathrm{EER}(c_j)$ 。则

$$\mathrm{DYE}(t) \;=\; \sum_{c \in \mathrm{children}(t)} \frac{N(c)}{N(t)} \cdot \mathrm{EER}(c).$$

（即按「到达概率」把子结点的错误率加权求和。）

5. 剪枝判定

若：

$$\mathrm{STE}(t) \;\le\; \mathrm{DYE}(t)$$

则把 t 剪为叶；否则保留子树。这个判断在自底向上遍历中对每个内部结点做一次。

6. 举例

很蒙？没事这个算法不属于重点记忆，因为不常用，但是不耽误我举例子（

现在：取二分类、先验均匀 $p_A = p_B = 0.5$ ，参数 $m = 1$ 。构造一个非常小的子树：结点 t 有两个子结点 L 与 R ，每个子结点都只有 1 个训练样本，且这两个样本都属于类 B（即两侧都是纯 B，不是骂你的，是骂我的）。因此：

- 左子结点 L：$N(L) = 1$ ，类计数 $n_B(L) = 1,\ n_A(L) = 0$ 。
- 右子结点 R：$N(R) = 1$ ，类计数 $n_B(R) = 1,\ n_A(R) = 0$ 。
- 父结点 t：$N(t) = 2$ ，类计数 $n_B(t) = 2,\ n_A(t) = 0$ 。

按步骤计算：

1. **对子结点 L（同样适用于 R）计算 $\hat{p}$ ：**

$$P_B(L) = \frac{n_B(L) + m \cdot p_B}{N(L) + m} = \frac{1 + 1 \cdot 0.5}{1 + 1} = \frac{1.5}{2} = 0.75.$$

所以子结点 L 的期望错误率：

$$\mathrm{EER}(L) = 1 - \hat{p}_{\max}(L) = 1 - 0.75 = 0.25.$$

（右子结点 R 同样是 0.25。）

2. **计算父结点 t 的动态错误率 DYE（按到达概率加权）：**

$$\mathrm{DYE}(t) = \frac{N(L)}{N(t)}\mathrm{EER}(L) + \frac{N(R)}{N(t)}\mathrm{EER}(R) = \frac{1}{2}\cdot 0.25 + \frac{1}{2}\cdot 0.25 = 0.25.$$

3. **计算父结点 t 的静态错误率 STE（作为单叶的错误率）：**

父结点多数类是 B，先验平滑后

$$\hat{p}_B(t) = \frac{n_B(t) + m\cdot p_B}{N(t)+m} = \frac{2 + 1\cdot 0.5}{2+1} = \frac{2.5}{3} \approx 0.833333\ldots$$

因此

$$\mathrm{STE}(t) = 1 - \hat{p}_{\max}(t) = 1 - 0.833333\ldots \approx 0.166666\ldots$$

4. **比较并决定：**

$$\mathrm{STE}(t) \approx 0.1667 \text{ 比 } \mathrm{DYE}(t) = 0.25 \text{ 更小},$$

所以按 MEP **应当剪掉** t 的子树，把 t 替换为单个叶（预测 B），因为这样期望错误率更小。

这个例子说明了，***当子结点样本量极小、且父结点整体多数类更“稳”时（使用 m-平滑后），MEP 会倾向于把不稳定的分支合并回父节点***。以上数值与 MEP 的公式一致（参考 MEP 的 m-estimate 与静、动误差定义）。

同样老规矩，论文贴出来：Niblett, T., & Bratko, I. (1986). *Learning decision rules in noisy domains.* Expert Systems.

可预见的是 ***MEP 在某些数据集上比 Pessimistic Error Pruning（PEP）更保守、剪枝更少，有时会产生更大的树且不一定更准确（不同文献中评估不一）***。因此在实践中常把 MEP 当成备选而非必用方法。(lamarr-institute.org, research.ijais.org)

Cost-Complexity Pruning (CCP, 代价复杂度剪枝)

这是最常用的默认的一个算法。**代价复杂度剪枝的核心思想**是：我们不单纯追求决策树在训练集上的低错误率，而是寻求一个***错误率与复杂度之间的平衡***。它认为，一棵树的“总代价”由两部分组成：

1. **预测错误带来的代价**（即误分类成本）。
2. **模型复杂度带来的代价。**

CCP通过一个参数 α 来平衡这两者。

*CCP定义了一个衡量子树整体“代价”的指标，称为**代价复杂度函数：***

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|$$

其中：

- $R(T)$ 是子树 T 在训练集上的**误分类成本**（对于分类树）或**平方误差**（对于回归树）。可以简单理解为子树在训练集上的错误率。
- $|\widetilde{T}|$ 是子树 T 的**叶节点数量**，代表了模型的复杂度。叶节点越多，模型越复杂。
- α (**复杂度参数**, $\alpha \geq 0$) 是平衡两者权重的系数。它代表了**每增加一个叶节点所需要付出的代价**。
  - $\alpha = 0$：只关心训练误差，不惩罚复杂度，即保留最大的树。
  - α 增大：对复杂度的惩罚力度加大，倾向于选择更简单的树。

剪枝的目标就是找到一棵使 $R_\alpha(T)$ 最小的子树。

1. 构建决策树

首先，使用训练数据构建一棵尽可能大的决策树 T_0 （直到所有叶节点都纯或达到最小节点样本数）。这棵树是剪枝的起点。

2. 计算节点的“强度”

对于树中的每个**非叶节点（内部节点）** t ，我们计算一个关键指标：**有效 α 或 阈值 α_t** 。这个 α_t 的含义是：**剪掉以节点 t 为根的子树 T_t ，需要付出的“单位复杂度代价”至少是多少才是值得的**。

计算公式为：

$$\alpha_t = \frac{R(t) - R(T_t)}{|\widetilde{T_t}| - 1}$$

公式推导与解释：

- $R(t)$：如果剪枝，即把节点 t 变为一个叶节点，该节点的误分类成本。
- $R(T_t)$：如果不剪枝，保留以 t 为根的整个子树 T_t 的误分类成本。
- $|\widetilde{T_t}|$：子树 T_t 的叶节点数量。
- **分子** $R(t) - R(T_t)$：剪枝带来的**错误成本增加**（因为剪枝后模型变简单，训练误差通常会上升）。
- **分母** $|\widetilde{T_t}| - 1$：剪枝带来的**复杂度降低**（剪枝后，用一个叶节点 t 替换了原本的 $|\widetilde{T_t}|$ 个叶节点，所以减少了 $|\widetilde{T_t}| - 1$ 个叶节点）。

所以，α_t 可以理解为 **“每减少一个叶节点，所允许的错误率增长”**。*α_t 越小，说明剪掉这个子树所付出的错误代价相对较小，收益（复杂度降低）更大，这个节点就越“弱”，越应该被剪掉*。

3. 剪掉最弱的链接

在当前树中，找到**最小有效 α** 对应的节点，即 $g(t) = \alpha_{min}$ 。将所有有效 α 为 α_{min} 的节点都剪掉（将其子树替换为叶节点）。

这样，我们就得到了一棵新的、更小的树 T_1 。

4. 迭代

对最新生成的树 T_1 ，重复步骤 2 和 3，计算其内部节点的有效 α ，并剪掉其中最弱的节点，得到 T_2 。如此反复，直到只剩下根节点，最终得到一系列嵌套的子树序列： $T_0, T_1, T_2, ..., T_k$ （其中 T_0 是最大树， T_k 是根节点）。

CCP的优点在于它提供了一个严格的、基于统计学的框架来平衡模型的偏差和方差。虽然计算量较大，但其剪枝效果通常非常好，是实践中常用且可靠的剪枝方法。

论文：Breiman, L., Friedman, J., Olshen, R.A., & Stone, C.J. (1984). *Classification and Regression Trees (1st ed.).* Chapman and Hall/CRC.

到此，我们介绍了四种常见的后剪枝算法。***一般情况下 CCP > REP > PEP > MEP（从实际应用效果角度看），但如果样本量少、没法划验证集，就用 PEP/MEP***。

References

[1] [History of pruning algorithm development and python implementation(finished)](https://blog.csdn.net/appleyuchi/article/details/83692381)

其余见文章内部