河北师范大学软件学院
Software College of Hebei Normal University

决策树剪枝

决策树剪枝

1. 预剪枝（前剪枝）

   1）通过提前停止树的构造来对决策树进行剪枝

   2）一旦停止该节点下树的继续构造，该节点就成了叶节点。

   3）该叶节点持有其数据集中样本最多的类或者其概率分布

2. 后剪枝

   1）首先构造完整的决策树，允许决策树过度拟合训练数据，

   2）然后对那些置信度不够的结点的子树用叶结点来替代

   3）该叶节点持有其子树的数据集中样本最多的类或者其概率分布

预剪枝

预剪枝方法可以归纳为以下几种：

1. 最为简单的方法就是在决策树到达一定高度的情况下就停止树的生长；
2. 到达此结点的样本数小于某一个阈值也可停止树的生长；
3. 该节点的样本具有相同的特征，而不必一定属于同一类，也可停止生长。可以处理样本集中的数据冲突问题；
4. 如果在最好情况下的信息(基尼系数)增益都小于某个阈值，即使有些样本不属于同一类，也停止树的生长。

后剪枝

1. 降低错误剪枝 REP (Reduced Error Pruning)
2. 代价-复杂度剪枝 CCP (Cost-Complexity Pruning)
3. 最小错误剪枝 MEP (Minimum Error Pruning)
4. 悲观错误剪枝 PEP (Pessimistic Error Pruning)
5. 基于错误剪枝 EBP (Error-Based Pruning)

降低错误剪枝REP(Reduced Error Pruning)

1. 独立的剪枝集D
2. 基本思路:
   a) 对于决策树 T 的每棵子树 s (非叶子节点), 用叶子替代这棵子树;
   b) 如果 s 被叶子替代后形成的新树关于 D 的误差等于或小于 s 关于 D 所产生的误差, 则用叶子替代子树 s.
3. 优点:
   a) 计算复杂性低;
   b) 对未知示例预测偏差较小.

代价-复杂度剪枝CCP(Cost-Complexity Pruning)

1. CCP又叫CART剪枝法
2. 代价(cost)
   a) 样本错分率；　b）样本条件熵.
3. 复杂度(complexity)：Briman定义树T代价复杂度
   a) 树t的叶节点数

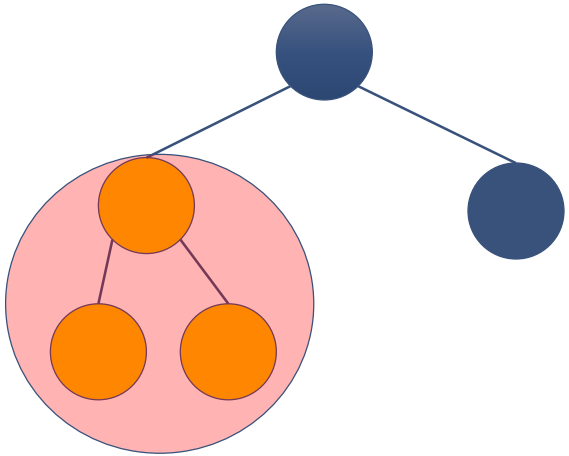
$$cc(T) = \frac{Ne}{N} + \alpha * |T|$$

其中，N 是决策树训练样本个数，Ne 是决策树 T 错分的样本数，|T| 是决策树 T 的叶子节点数量

1. 参数α：用于衡量代价与复杂度之间关系
   a) 表示剪枝后树的复杂度降低程度与代价间的关系
   b) 如何确定α的值
      ① （自动求，手动设置）

如何自动设置α的值：

对 t 来说，剪掉它的子树 s，
以 t 中最优叶节点代替，
得到新树 new_t。
new_t 比 t 对于训练数据分错M个，
但是 new_t 叶节点数比 t 少(|s|- 1)个



1. 令替换前后的代价复杂度相同

$$cc(t) = cc(new\_t)$$

$$=> \frac{E}{N} + \alpha * |t| = \frac{E+M}{N} + \alpha * [|t| - (|s| - 1)]$$

$$=> \alpha = \frac{M}{N(|s| - 1)}$$

其中，M是用叶节点替换 t 子树以后，增加的错分样本数。

|s| 是子树 s 的叶节点数

CCP剪枝步骤：（不需要手动设置α）

第一步：

计算完全决策树T的每个非叶节点的α值；

循环剪掉具有最小α值的子树，直到剩下根节点；

得到一系列剪枝(嵌套)树 $\{T_0, T_1, \cdots, T_m\}$ 其中 T_0 就是完全决策树 T ，

T_{i+1} 是对 T_i 进行剪枝得到的结果

第二步：

使用独立的验证集(非训练集)对第一步中的 T_i 进行评估，获取最佳剪枝树

标准错误SE(standart error)，公式：

$$SE(Ne') = \sqrt{\frac{E_N' * (N' - E_N')}{(N')^2}}$$

其中： N' 是验证集的样本数目；Ne_i' 表示树 T_i 对验证集的错分样本数；

令： $E_N' = \min\{E_{Ni}'\}$

最佳剪枝树：T_best 是满足以下条件并且包含的节点数最少的那颗剪枝树。

$$E_i \le E_N' + SE(E_N') \quad and \quad \min\{|t_i|\}$$

优点：剪枝之后的决策树在具有较好的稳定性(错误率提高的不多)
剪枝后的树要简单(使用模型时，时间会减少)
不需要手动设置alpha的值(alpha是根据不同树自动调整的)
缺点：计算的时间复杂度，空间复杂度都很高

正则剪枝：（手动设置α）

输入：整个树 T，参数α

输出：修剪后的 T_α

1. 计算每个节点的错误数(包括页节点)
2. 递归的从树的叶节点回缩，设一组叶节点回缩到其父节点之前和之后分别是：T_A 和 T_B，其对应的错误样本数分别是：$C_\alpha(T_A)$和$C_\alpha(T_B)$

   如果：

$$C_\alpha(T_A) \leq C_\alpha(T_B)$$

   则进行剪枝，即将父节点更新为叶子节点

更换代价公式

更换代价公式：
正则化公式由两部分构成：预测误差和树复杂度

用熵来代替错误率：
假设：树 T 的叶节点个数 $|T|$，t 是树 T 的叶节点，N_t 表示该叶节点的样本个数，其中 k 类样本有 N_{tk} 个
损失函数定义为：

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha |T|$$

其中

$$H(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

最小错误剪枝MEP(Minimum Error Pruning)

1. 基本思路：
   a) 采用自底向上的方式，对于树中每个非叶节点，首先计算该节点的误差E(t)；
   b) 计算该节点每个分枝的误差，并且加权相加，权为每个分枝拥有的训练样本比例；
   c) 如果 E(t)大于分枝误差期望，则保留该子树；否则，剪裁它。
2. 不需要单独的验证集，只需要使用训练集就可以；
3. 效果很一般，而且不够稳定

悲观错误剪枝PEP(Pessimistic Error Pruning)

1. 克服REP需要独立剪枝集的缺点
2. 误差估计的连续性校正
3. 自上而下
4. 缺点(悲观)：
   基于训练集建立的树，对训练集合的错误率，对于未知集合来说是不可信的
5. 重点：**经验惩罚因子**；**标准错误**

悲观错误剪枝PEP(Pessimistic Error Pruning)

1. 设原始决策树 T，叶节点为 t，t 节点训练实例个数为 N_t，其中错分个数为 E_{Nt}
2. 错误率定义为：$p(E_{Nt}) = E_{Nt} / N_t$
3. 增加连续性校正：$p(E_{Nt}) = (E_{Nt} + 0.5) / N_t$
4. 对应的错分样本数：$E_{Nt} + 0.5$
5. 增加连续性校正后：所有子树的错分数 $E_t = \sum E_s + |s| / 2$

悲观错误剪枝PEP(Pessimistic Error Pruning)

6. 标准误差：

$$SE(E_t)=\sqrt{\frac{E_t*(N_t-E_t)}{N_t^{\,2}}}$$

其中，N_t 是当前训练数据量，E_t 是对校正后子树上的数据错分数。

7. 剪枝条件：

$$(E+0.5)<E_t+SE(E_t)$$

其中，E 是当前的数据错分数

悲观错误剪枝PEP(Pessimistic Error Pruning)

悲观剪枝的本质：
模型的(判断对，判断错)是满足**伯努利分布**

剪枝的条件的本质：

根据置信区间，我们设定一定的显著性因子，我们可以估算出误判次数的上下界。 **统计检验**

模型的判断能力还可以看做的**正态分布**

基于错误剪枝EBP(Error-Based Pruning)

更加悲观(对其他的数据集效果更不稳定)

不需要验证集

自下而上的剪枝

改进了基于悲观错误剪枝的方法：

提出了嫁接的思想

除了比较该决策点与子树的校正错误率，还会比较该决策节点下最大的子树的校正错误率。

基于错误剪枝EBP(Error-Based Pruning)

- 第一步：计算叶节点的错分样本率估计的置信区间上限U
- 第二步：计算叶节点的预测错分样本数
  - 叶节点的预测错分样本数=到达该叶节点的样本数*该叶节点的预测错分样本率U
- 第三步：判断是否剪枝及如何剪枝
  - 分别计算三种预测错分样本数：
    - 计算子树t的所有叶节点预测错分样本数之和，记为E1
    - 计算子树t被剪枝以叶节点代替时的预测错分样本数，记为E2
    - **计算子树t的最大分枝的预测错分样本数，记为E3**
  - 比较E1，E2，E3，如下：
    - E1最小时，不剪枝
    - E2最小时，进行剪枝，以一个叶节点代替t
    - **E3最小时，采用“嫁接”(grafting)策略，即用这个最大分枝代替t**

降低错误剪枝 REP 实现过程

计算重点：

利用tree对test_data进行**划分**

计算剪枝前的错误率:(左右子树错误率加权和)

计算剪枝后的错误率:

求剪枝后的标签

如何剪枝:**(塌陷处理)**

如果测试集为空：无论是不是叶节点直接塌陷

如果测试集不为空：从叶节点开始进行塌陷

降低错误剪枝 REP 实现过程

需要的函数：

1. data_split(tree,data)：
   根据树桩的决策将data进行划分
2. get_label(tree)：
   根据树桩位置的决策点得到塌陷后的标签
3. pruning_CART(tree,test_data)：
   对树进行塌陷处理
4. is_tree(obj)：
   判断obj是否是树；判断是否到达叶子节点

降低错误剪枝 REP 实现过程

输入：带groups的tree;剪枝集 data(test_data)

输出：pruning之后的tree