4. 决策树

决策树是一类用树结构定义决策函数/决策空间的划分的算法。决策树的叶节点代表分类后所获得的分类标记/或预测值，决策树的非叶子节点以及其生出的分支代表划分准则，目前比较流行以二分树作为决策树的结构约束。

构建步骤：

特征选择-决策树生成-决策树剪枝

4.1 基本概念

信息熵

平均而言发生的一个时间我们得到的信息量的大小。（信息量的期望）熵越大，样本的不确定性越大。

随机变量的信息熵：pi：每个xi发生的概率

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i$$

对于样本集合D，有K个类别，|Ck|是类别k的样本个数，|D|是样本总数：

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$$

信息增益

信息增益是以某特征划分数据集前后的熵的差值，可以用某特征划分数据集前后的熵的差值来衡量此特征对于样本集合D划分效果的好坏。给定特征A与训练数据集D,定义信息增益g(D,A)为：

$$g(D, A) = H(D) - H(D|A) = H(D) - \sum_{i=1}^{n} \frac{N_i}{N} \sum_{k=1}^{K} -(\frac{N_{ik}}{N_i} \log \frac{N_{ik}}{N_i})$$

信息增益比

可以通过定义信息增益比来解决信息增益向取值较多的特征的问题。

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)} \quad H_A(D) = -\sum_{i=1}^{n} \frac{N_i}{N} \log \frac{N_i}{N}$$

基尼系数

基尼指数表示在样本中一个随机选中的样本被分错的概率。Gini指数越小表示集合中被选中的样本被分错的概率越小，也就是说集合的纯度越高，反之，集合越不纯。

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

4.2 决策树生成

熵和基尼系数都是针对于分类，方差针对于回归。

ID3算法

从根节点开始，对节点计算所有可能的信息增益，选择最大的信息增益特征作为节点的特征，建立子节点，再对子节点递归调用以上方法，直到信息增益很小为止。

算法 5.2（ID3 算法）

输入：训练数据集 D，特征集 A，阈值 ε；

输出：决策树 T.

（1）若 D 中所有实例属于同一类 C_k，则 T 为单结点树，并将类 C_k 作为该结点的类标记，返回 T；

（2）若 $A=\varnothing$，则 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T；

（3）否则，按算法 5.1 计算 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g；

（4）如果 A_g 的信息增益小于阈值 ε，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T；

（5）否则，对 A_g 的每一可能值 a_i，依 $A_g=a_i$ 将 D 分割为若干非空子集 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由结点及其子结点构成树 T，返回 T；

（6）对第 i 个子结点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用步 (1) ～步 (5)，得到子树 T_i，返回 T_i. ■

只有树的生成，容易产生过拟合。

ID4.5算法

与ID3算法类似，但是采用信息增益比选择特征。

算法 5.3（C4.5 的生成算法）

输入：训练数据集 D，特征集 A，阈值 ε；

输出：决策树 T.

（1）如果 D 中所有实例属于同一类 C_k，则置 T 为单结点树，并将 C_k 作为该结点的类，返回 T；

（2）如果 $A=\varnothing$，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类，返回 T；

（3）否则，按式(5.10)计算 A 中各特征对 D 的信息增益比，选择信息增益比最大的特征 A_g；

（4）如果 A_g 的信息增益比小于阈值 ε，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类，返回 T；

（5）否则，对 A_g 的每一可能值 a_i，依 $A_g=a_i$ 将 D 分割为子集若干非空 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由结点及其子结点构成树 T，返回 T；

（6）对结点 i，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用步(1)～步(5)，得到子树 T_i，返回 T_i. ■

缺点是在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。此外。C4.5只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

4.3 决策树剪枝

分为先剪枝和后剪枝两种。先剪枝即提前结束决策树的增长，与上述决策树停止生长的方式一样。后剪枝是指决策树生长完成后再进行剪枝的过程。

剪枝通过极小化决策树整体损失函数实现，树节点个数为T，t是树的叶子节点，叶节点有N_t个样本点，其中k类样本点有N_tk个，H(T)为再t上的信息熵，损失函数：

$$C_a(T)=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha|T|=-\sum_{t=1}^{|T|}\sum_{k=1}^{K}N_{tk}\log\frac{N_{tk}}{N_t}+\alpha|T|=C(T)+\alpha|T|$$

C(T)表示模型对训练数据的预测误差，即模型与训练数据的拟合程度，|T|表示模型复杂程度，参数α控制两者之间的影响。

算法 5.4（树的剪枝算法）

输入：生成算法产生的整个树 T，参数 α；

输出：修剪后的子树 T_α.

（1）计算每个结点的经验熵.

（2）递归地从树的叶结点向上回缩.

设一组叶结点回缩到其父结点之前与之后的整体树分别为 T_B 与 T_A，其对应的损失函数值分别是 $C_\alpha(T_B)$ 与 $C_\alpha(T_A)$，如果

$$C_\alpha(T_A) \leqslant C_\alpha(T_B) \tag{5.15}$$

则进行剪枝，即将父结点变为新的叶结点.

（3）返回 (2)，直至不能继续为止，得到损失函数最小的子树 T_α. ■

CART算法

分类与回归树是决策树学习算法之一。同样由特征选择、生成、剪枝组成，决策树为二叉树。

CART生成

回归树：

用平方误差最小化准则生成，算法如下：

算法 5.5（最小二乘回归树生成算法）

输入：训练数据集 D；

输出：回归树 $f(x)$.

在训练数据集所在的输入空间中，递归地将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

（1）选择最优切分变量 j 与切分点 s，求解

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{5.21}$$

遍历变量 j，对固定的切分变量 j 扫描切分点 s，选择使式 (5.21) 达到最小值的对 (j,s). **选取第j个变量和取值s作为切分变量与切分点**

（2）用选定的对 (j,s) 划分区域并决定相应的输出值：

$$R_1(j,s)=\{x\mid x^{(j)}\leqslant s\},\quad R_2(j,s)=\{x\mid x^{(j)}>s\}$$

$$\hat{c}_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)}y_i,\quad x\in R_m,\quad m=1,2$$

（3）继续对两个子区域调用步骤 (1)，(2)，直至满足停止条件.

（4）将输入空间划分为 M 个区域 $R_1,R_2,\cdots,R_M$，生成决策树：

$$f(x)=\sum_{m=1}^{M}\hat{c}_m I(x\in R_m)$$ ■

分类树:

用基尼指数选择最优特征，同时决定该特征的最优二值切分点

算法 5.6（CART 生成算法）

输入：训练数据集 D，停止计算的条件；

输出：CART 决策树.

根据训练数据集，从根结点开始，递归地对每个结点进行以下操作，构建二叉决策树：

（1）设结点的训练数据集为 D，计算现有特征对该数据集的基尼指数. 此时，对每一个特征 A，对其可能取的每个值 a，根据样本点对 $A=a$ 的测试为“是”或“否”将 D 分割成 D_1 和 D_2 两部分，利用式（5.25）计算 $A=a$ 时的基尼指数.

（2）在所有可能的特征 A 以及它们所有可能的切分点 a 中，选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点. 依最优特征与最优切分点，从现结点生成两个子结点，将训练数据集依特征分配到两个子结点中去.

（3）对两个子结点递归地调用（1），（2），直至满足停止条件.

（4）生成 CART 决策树. ■

CART剪枝

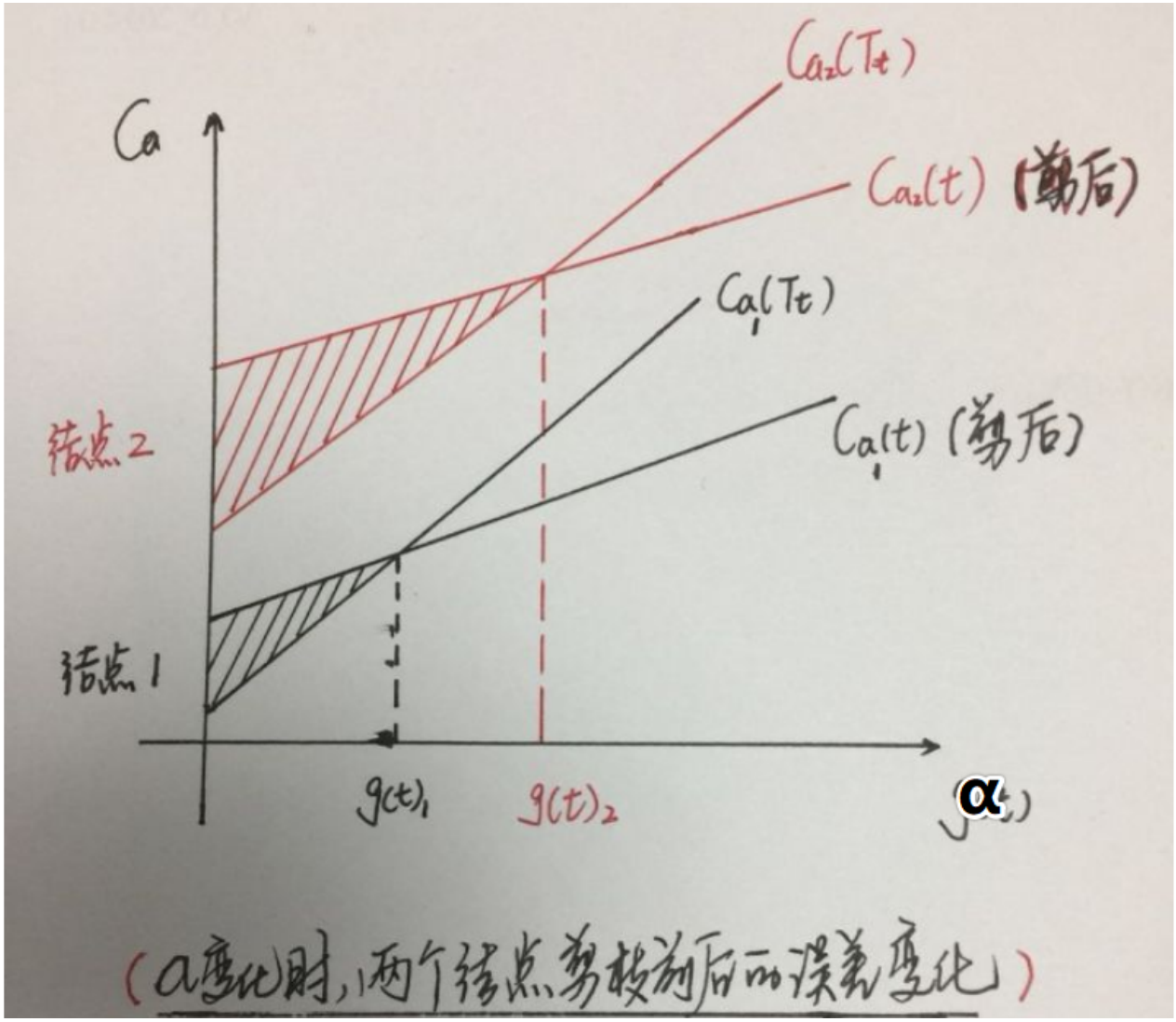
首先从生成算法产生的决策树T0底端开始剪枝，直到T0根节点，形成一个子树序列，然后通过交叉验证再独立的验证数据集，谁误差小即为最优子树。

https://blog.csdn.net/zhengzhenxian/article/details/79083643 这位大佬说的很明白，点赞。

g(t)表示剪枝后整体损失函数减少的程度，也就是剪枝的阈值。如果g(t)计算之后与总体损失函数中的参数α相等，T_t与t有相同的损失函数值，剪和不剪总体函数一样。如果α更大，不剪的总体函数是大于剪去的，即剪枝后会使总体函数变小，这样就可以对T_t剪枝，反之不用。

这里面涉及到一个定理：将α从小增大，0=α0<α1<α2...<αn，在区间[αi,αi+1)中，子树Ti是这个区间里面最优的。当alpha从0开始缓慢增大，总会有某棵子树该剪，其他子树不该剪的情况，即alpha超过了某个结点的g(t)，但还没有超过其他结点的g(t)。这样随着alpha不断增大，不断地剪枝，就得到了n+1棵子树，接下来只要用独立数据集测试这n+1棵子树，试试哪棵子树的误差最小就知道那棵是最好的方案了。

为什么要选择最小的g(t)呢？理解如下：



以图中两个点为例，结点1和结点2，g(t)2大于g(t)1，假设在所有结点中g(t)1最小，g(t)2最大，两种选择方法：当选择最大值g(t)2，即结点2进行剪枝，但此时结点1的不修剪的误差大于修剪之后的误差，即如果不修剪的话，误差变大，依次类推，对其它所有的结点的g(t)都是如此，从而造成整体的累计误差更大。反之，如果选择最小值g(t)1，即结点1进行剪枝，则其余结点不剪的误差要小于剪后的误差，不修剪为好，且整体的误差最小。从而以最小g(t)剪枝获得的子树是该alpha值下的最优子树！

算法 5.7（CART 剪枝算法）

输入：CART 算法生成的决策树 T_0；

输出：最优决策树 T_α.

（1）设 $k=0$，$T=T_0$.

（2）设 $\alpha=+\infty$.

（3）自下而上地对各内部结点 t 计算 $C(T_t)$，$|T_t|$ 以及

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

$$\alpha=\min(\alpha,g(t))$$

这里，T_t 表示以 t 为根结点的子树，$C(T_t)$ 是对训练数据的预测误差，$|T_t|$ 是 T_t 的叶结点个数.

（4）自上而下地访问内部结点 t，如果有 $g(t)=\alpha$，进行剪枝，并对叶结点 t 以多数表决法决定其类，得到树 T.

（5）设 $k=k+1$，$\alpha_k=\alpha$，$T_k=T$.

（6）如果 T 不是由根结点单独构成的树，则回到步骤 (4).

（7）采用交叉验证法在子树序列 $T_0,T_1,\cdots,T_n$ 中选取最优子树 T_α. ■