决策树

ID3树与C4.5树

假设有数据集$D=\{(x_i,y_i)\}$，记其特征数为n， 数据集大小为$|D|$。我们希望构建一系列规则R_i，将数据集分成若干块，其中任意一块数据集中的y都相同，其中每条规则R_i都只针对数据集的一个特征。这种分法自然地引导我们考虑树形结构，其中根节点中包含所有的数据，每一个父节点到子节点的分割代表一个分割规则R_i，子节点中的数据是父节点的数据中满足规则R_i的部分。

那么在每次分割时，如何确定相应规则呢？我们需要一个指标来评估规则的优劣。在感觉上，我们希望将数据分的越“开”越好，所以我们引入物理学中熵（entropy）（记为H）的概念，如果D有K类，那么$H(D)$定义为：

$$H(D)=-\sum_{k=1}^{K}\frac{|D_k|}{|D|}\log_2\frac{|D_k|}{|D|}=-\sum_{k=1}^{K}p_k\log_2 p_k$$

其中p_k就是从D中随机抽取一个观测，观测为类k的概率。其中p_k的分布越不均匀，$H(D)$越小。

假设我们的规则是特征i的不同取值，即：假设特征i的去重取值为$x_{1i},x_{2i},\dots,x_{Mi}$，将特征$i$的取值为$x_{1i}$的数据放到第1个子节点，取值为$x_{2i}$的数据放到第2个子节点，以此类推，取值为$x_{Mi}$的数据放到第$M$个子节点。那么在给定该规则下，熵$H(D|i)$为各个子节点的熵的加权平均：

$$H(D|i)=\sum_{m=1}^{M}\frac{|D_m|}{|D|}H(D_m)$$

显然，由于我们获取了“特征i的取值为x_{mi}”这一信息，则条件熵一定会比非条件熵要小，我们可以定义信息增益(info gain)$g(D,i)=H(D)-H(D|i)$，信息增益越大，说明该规则的分类效果越好，即

$$i_{best}=\arg\max_i \quad g(D,i)$$

然后递归让树生长即可。当使用信息增益作为衡量指标时，称此树为ID3决策树。

但是ID3决策树有明显的缺点，即更容易青睐去重取值较多的特征。比如如果特征i有N个不同取值，则每个子节点中只有一条观测，每个节点的熵自然为0，条件熵也为0，信息增益自然最大。我们为了克服这一问题，希望将信息增益归一化，即考虑到特征i的去重取值数。所以我们引入特征i的固有值(inherent value)的概念，即将特征i的每个去重取值看做一个类：

$$IV(D,i)=-\sum_{m=1}^{M}\frac{|D_m|}{|D|}\log_2\frac{|D_m|}{|D|}$$

然后用固有值去修正信息增益，定义信息增益率(info gain ratio)为：

$$gr(D,i)=\frac{g(D,i)}{IV(D,i)}=\frac{H(D)-H(D|i)}{IV(D,i)}$$

选取最优的特征i为：

$$i_{best}=\arg\max_i \quad gr(D,i)$$

当然，用信息增益率作为指标，会更偏向取值较少的指标。所以在挑选最优特征的时候，还要加上一个限定条件：参与比较的特征，其信息增益要高于平均值，即

$$i_{best} = \arg\max_i \quad gr(D,i)$$

$$s.t. \quad g(D,i) > \frac{\sum_{i=1}^{n} g(D,i)}{n}$$

以以上准则得到的树称为C4.5决策树。

注意到，在ID3和C4.5决策树中，每个特征只会使用一次。

ID3决策树的生成算法为：

1）读取数据集D，初始化决策树

2）按照准则$i_{best} = \arg\max_i \quad g(D,i)$选取分割的特征

3）对i_{best}的每个去重取值x_{im}，将特征$i_{best} = x_{im}$的数据集删除该特征后，作为树的第m个子节点的数据集

4）对每个子节点递归分割，直到熵降为0(所有数据都是同一类)，或数据集只剩下1个特征为止

5）对停止分割的叶子节点，设置叶子节点的标签为节点中出现频率最高的取值

C4.5决策树的生成算法为：

1）读取数据集D，初始化决策树

2）按照准则$i_{best} = \arg\max_i \quad gr(D,i) \quad s.t. \quad g(D,i) > \frac{\sum_{i=1}^{n} g(D,i)}{n}$选取分割的特征

3）对i_{best}的每个去重取值x_{im}，将特征$i_{best} = x_{im}$的数据集删除该特征后，作为树的第m个子节点的数据集

4）对每个子节点递归分割，直到熵降为0(所有数据都是同一类)，或数据集只剩下1个特征为止

5）对停止分割的叶子节点，设置叶子节点的标签为节点中出现频率最高的标签值

显然，ID3树与C4.5树的深度不会超过$n-1$，每个节点的子节点数是待分割特征的去重取值数。

使用测试集预测时，按照节点的特征和规则递归地从根节点下降到叶子节点，然后返回叶子节点的标签值即可。如果测试集在特征i上的取值与已有的子节点均不同，则随机返回一个子节点。

ID3树与C4.5树的剪枝

决策树有很高的过拟合的风险。为了防止过拟合，我们需要剪掉决策树的一些非叶子节点。但怎么选择剪掉哪些非叶子节点呢?

假设我们已经得到了决策树T，定义t是T中的任一非叶子节点，$|T|$是T的子节点中叶子节点的数目，α是正则因子，则定义t的原损失函数为：

$$C_0(t) = \sum_{j=1}^{|T|} \frac{|D_{tj}|}{|D_t|} H(D_{tj}) + \alpha|t|$$

其中D_{tj}是t的第j个叶子节点中的数据集。

t的剪枝后损失函数为：

$$C_1(t) = H(D_t) + \alpha \cdot 1$$

剪枝后，t本身将退化为叶节点，D_t是t的根节点中的数据集。

如果$C_1(t) \leq C_0(t)$，则对t执行剪枝，即删除其所有子节点，并将t的标签设为D_t中出现频率最高的标签值。

剪枝算法如下：

1）读取完整的决策树T与正则因子α，初始化一个存储待检查节点的栈S

2）如果T本身就是叶子节点，则无需剪枝，结束程序；否则将T的根节点压入S

3）取S顶部的节点t，对t的所有子节点，判断其是否为叶节点或被检查过的非叶子节点

4-1）如果t的所有子节点都是叶子节点或所有子节点都已经被检查过，则将t出栈，计算$C_0(t), C_1(t)$。如果$C_1(t) \leq C_0(t)$，则剪枝；否则将t标记为已检查过

4-2）如果t的子节点中，有未被检查过的非叶子节点，则将这些子节点都压入S

5）循环，直到S为空为止。返回剪枝后的T

注意：

1）S中不可能有叶子节点，这是步骤2与步骤4-2所保证的；

2）步骤4-2时，t并不出栈，也不用判断是否需要对其剪枝。因为t还有可能被剪枝的子节点，如果子节点被剪枝了，t的原损失函数需要重新计算，所以等到其所有子节点都是叶节点或是已检查过的非叶子节点时，再判断t是否需要被剪枝即可。

CART树

注意到ID3树与C4.5树有以下3个不足之处：

1）ID3树与C4.5树都是多叉树，在决策树生成过程中会产生较多的节点；

2）ID3树与C4.5树以熵作为分割准则，求熵时会需要大量实际生产时比较耗时的log运算；

3）ID3树与C4.5树的正则因子α难以确定，需要靠经验摸索。

据此，CART（Classifier And Regression Tree）树应运而生。它采用二叉树的结构，解决了节点较多的问题；采用基尼系数作为分类准则，避免了对数运算；采用独特的剪枝方法，直接可以确定最优的正则因子α。

首先介绍基尼系数：如果D有K类，那么$Gini(D)$定义为：

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|D_k|}{|D|} \right)^2 = 1 - \sum_{k=1}^{K} p_k^2$$

其中p_k就是从D中随机抽取一个观测，观测为类k的概率。其中p_k的分布越不均匀，$Gini(D)$越小。

对每一个特征i的每一个去重取值x_{im}，定义条件基尼系数为

$$Gini(D|i, x_{im}) = \frac{|D|X_i = x_{im}|}{|D|} Gini(D_{i,x_{im}}) + \frac{|D|X_i \neq x_{im}|}{|D|} Gini(D_{i,!x_{im}})$$

其中$D_{i,x_{im}}$代表特征i为x_{im}的子数据集，$D_{i,!x_{im}}$代表特征i不为x_{im}的子数据集。

迭代每个特征与特征的每个去重取值，得到最优的特征与分割点：

$$i_{best}, m_{best} = \arg\min_{i,m} \quad Gini(D|i, x_{im})$$

注意，这里与ID3树与C4.5树有所区别的是，这里直接比较条件基尼系数，而不是“基尼系数增益”（其实二者等价，因为非条件基尼系数都是一样的，相应的，ID3树与C4.5树在实际实现时也可以只比较条件熵）；以及，CART树的特征可以多次使用，不像ID3树与C4.5树只能使用一次。

找到分割点后，将$D_{i,x_{im}}$作为左子节点的数据集，$D_{i,!x_{im}}$作为右子节点的数据集继续分割即可。

CART分类决策树的生成算法为：

1）读取数据集D，初始化决策树

2）按照准则$i_{best}, m_{best} = \arg\min_{i,m} \quad Gini(D|i, x_{im})$选取分割的特征与相应的值

3）将$D_{i,x_{im}}$作为左子节点的数据集，$D_{i,!x_{im}}$作为右子节点的数据集继续递归分割，直到基尼系数降为0(所有数据都是同一类)为止

4）对停止分割的叶子节点，设置叶子节点的标签为节点中出现频率最高的取值

CART树的另一个优势是，他可以推广到解决回归问题，即用平方误差取代基尼系数作为分割准则：

$$Square(D) = \sum_{i=1}^{N} (y_i - \bar{y})^2 \equiv var(y) * (N-1)$$

平方误差越小，说明回归树的效果越好。

对每一个特征i的每一个去重取值x_{im}，定义条件平方误差为

$$Square(D|i, x_{im}) = |D|X_i \leq x_{im}| \cdot Square(D_{i,\leq x_{im}}) + |D|X_i > x_{im}| \cdot Square(D_{i,>x_{im}})$$

与分类决策树不同的是，回归决策树的分割是将数据分为特征$i \leq x_{im}$和$i > x_{im}$两部分。

最优的特征和分割点选取过程与分类树相同：

$$i_{best}, m_{best} = \arg\min_{i,m} \quad Square(D|i, x_{im})$$

此外，由于回归树的叶子节点的y不太可能是完全相同的值，所以我们一般会设置一个最小平方误差值δ，当$Square(D) < \delta$时，则不再继续分割数据，将该节点作为叶子节点，该节点y的均值作为该节点的值。

1）读取数据集D，初始化决策树

2）按照准则$i_{best}, m_{best} = \arg\min_{i,m} \quad Square(D|i, x_{im})$选取分割的特征与相应的值

3）将$D_{i,\leq x_{im}}$作为左子节点的数据集，$D_{i,>x_{im}}$作为右子节点的数据集继续递归分割，直到平方误差$< \delta$为止

4）对停止分割的叶子节点，设置叶子节点的值为节点中y的均值

使用测试集预测时，按照节点的特征和规则递归地从根节点下降到叶子节点，然后返回叶子节点的标签值（值）即可。

CART树的剪枝

CART树的剪枝与ID3树与C4.5树相比有所改良，不需要事先设定正则因子α。显然，由于数据集D是有限的，那么生成的决策树T也是有限的，不论α在$[0, +\infty)$的区间内如何取值，可能是最优的决策树的数量不会超过T中非叶子节点的个数，α越大，树的结构越简单，当$\alpha \to +\infty$时，最优决策树是根节点为叶节点的树。所以我们能通过比较有限个可能最优的决策树的预测效果，来在无穷区间中寻找最优的α。

首先，对决策树T任意的非叶子节点t，我们定义其原经验损失函数如下：

$$C_{C0}(t)=\sum_{j=1}^{|T|}\frac{|D_{tj}|}{|D_t|}Gini(D_{tj})+\alpha|t|$$

$$C_{R0}(t)=\sum_{j=1}^{|T|}Square(D_{tj})+\alpha|t|$$

其中下角标为C代表分类决策树，R代表回归决策树，其他定义与上文相同。

其剪枝经验损失函数如下：

$$C_{C1}(t)=Gini(D_t)+\alpha\cdot 1$$
$$C_{R1}(t)=Square(D_t)+\alpha\cdot 1$$

显然，当（第一个式子对应分类树，第二个式子对应回归树）不等式

$$\alpha\leq\frac{Gini(D_t)-\sum_{j=1}^{|T|}\frac{|D_{tj}|}{|D_t|}Gini(D_{tj})}{|t|-1}\left(\alpha\leq\frac{Square(D_t)-\sum_{j=1}^{|T|}Square(D_{tj})}{|t|-1}\right)$$

成立时，该叶子节点应当被剪枝。

于是我们对每个非叶子节点计算临界值α_t，取$t_{prune}=\arg\min_t\quad\alpha_t$并将其剪枝。原树$T$就是$\alpha\in[0,\min_t\alpha_t)$范围内的最优子树。

在剪枝得到的子树T'重复此步骤计算临界值$\min_t\alpha'_t$并剪枝，T'就是$\alpha\in[\min_t\alpha_t,\min_t\alpha'_t)$范围内的最优子树。

不断重复此步骤，直到剪枝后的树$T^{(\infty)}$的根节点即为叶节点为止，$T^{(\infty)}$就是$\alpha\in[\min_t\alpha_t^{(\infty)},+\infty)$范围内的最优子树。

于是我们得到了剪枝算法：

1）读取未剪枝的决策树$T^{(0)}$，如果$T^{(0)}$的根节点是叶节点，结束程序；记录循环次数k，建立空哈希表M，存放一系列最优的子树，其中键为α的区间，值为在此区间内的最优子树；最优区间下界α_m（每次循环计算出的是上界）

2）对$T^{(k)}$的每一个非叶子节点$t^{(k)}$，计算其临界值$\alpha_t^{(k)}$

3）排序得到最小临界值$\alpha_c^{(k)}=\arg\min_{t^{(k)}}\quad\alpha_t^{(k)}$与取到最小临界值的节点$t_c^{(k)}$，将该节点剪枝，得到子树$T^{(k+1)}$，记录$M[[\alpha_m,\alpha_c^{(k)})]=T^{(k)}$

4）$\alpha_m\to\alpha_c^{(k)}$，$k\to k+1$，判断$T^{(k)}$是否为根节点为叶节点的树，如果是，退出程序；否则继续2）

5）对程序停止之后的$T^{(k)}$，记录$M[[\alpha_m,+\infty)]=T^{(k)}$

6）取独立于D的验证集，将M中的一系列最优子树用验证集验证分类/回归效果（其中分类树用准确率/AUC验证；回归树用MSE/平方误差验证），选取表现最好的子树和对应的α的区间返回即可。

执行完该段算法后，就可以得到最优的剪枝子树和对应的最优α了（一个范围）。

Breiman等人证明，每次循环得到的$\alpha_c^{(k)}$一定是单调递增的序列，所以不必担心$\alpha_c^{(k+1)}\leq\alpha_c^{(k)}$，导致区间不存在的情况。

以上就是CART树剪枝的算法。当然，CART树也有缺点：

1）需要遍历每个特征的每个取值，比较耗时；

2）二叉树的深度往往要远深于ID3树与C4.5树的多叉树，但由于总节点数较少，所以问题不大；

3）需要额外的验证集来确定最优子树，但由于其他超参数（例如树的最大深度，总节点数等）也需要验证集来选择，所以问题也不大；

4）当解决分类问题时，对于分布比较分散的数据（即每个取值的频数都很小），二叉树会有接近一半的节点是叶节点（每个左节点的数据都很少甚至只有一条），这种树可能是极不平衡的，做预测时比较费时间。