决策树

决策树和随机森林

补充：

决策树防止过拟合的主要方法是剪枝，分预剪枝和后剪枝。但预剪枝容易欠拟合，因为很有可能在下一个节点分类精度又高了。所以主要采用后剪枝。

决策树优点：

可解释性强，是if-then的规则集合

预测速度快

需要较少的特征工程（不需要标准化，不需要处理连续值和缺失值）

连续值处理：首先将训练样本在当前这个连续特征的取值情况进行排序，每两个取值之间取平均得到一个可切分的点，此时已经和离散特征差不多了。

缺失值处理：XGBoost的时候再说

1.基本思想：

选取输入实例的每一维特征进行划分（属性测试），构造一棵树。这对应于人类的决策习惯，对选定的维度上的特征，满足则进入一个子节点，不满足则进入另一个子节点，最终找到叶节点（对应着决策结果，也就是类别）。其本质是从训练数据集中归纳出一组分类规则。

决策树学习的算法通常是递归地选择最优特征，并根据该特征对训练数据进行分割。对应决策树的构建过程：

a.构建根节点，存储所有的训练数据　b.选择最优特征，并按照该特征将训练数据集分割为子集，使得各个子集有一个在当前训练条件下最好的分类。如果这些子集已经能够被基本正确的分类，则构造叶节点　　c.如果还有子集不能被基本正确地分类，则继续选取最优特征，构建子节点。

这样构造出来的决策树对训练数据有很好的分类效果，但对测试数据却未必。这是因为可能发生了过拟合，需要进行剪枝。

决策树学习算法的三要素：特征选择，决策树的生成以及剪枝。

2.特征选择

信息熵：是随机变量的不确定性的度量。假设X是一个取有限个值的离散随机变量，且$P(X=xi)=pi$，$i=1,2,...,n$，则熵的定义为：

$H(X)=-\sum i=1npilogpi$，其中$0\leqslant H(x)\leqslant logn$

条件熵：表示在已知随机变量X的条件下Y的不确定性，定义为X给定条件下Y的条件熵对X的数学期望

$$H(Y|X)=\sum_{i=1}^{n}p_iH(Y|X=x_i)$$

其中，$p_i=P(X=x_i),i=1,2,...,n$

就是说通过选定的特征A对数据集进行划分，在每一个子集里都有一个信息熵。选定的特征有几个取值就有几个子集，条件上就是对子集上的信息熵取期望。

信息增益：定义为集合D的经验熵H(D)与特征A给定条件下D的经验熵H(D|A)之差，也成为互信息

g（D，A）=H(D)-H(D|A)

信息增益比：利用信息增益选择最优特征存在一个问题，就是选取规则偏向于取值比较多的特征。可以利用信息增益比来进行修正。

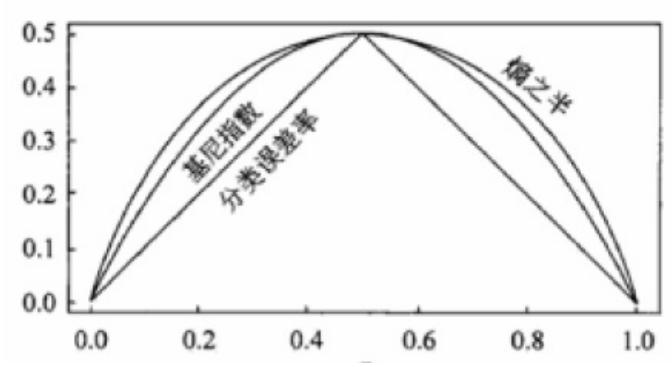
定义 5.3（信息增益比） 特征 A 对训练数据集 D 的信息增益比 $g_R(D,A)$ 定义为其信息增益 $g(D,A)$ 与训练数据集 D 的经验熵 $H(D)$ 之比：

$$g_R(D,A)=\frac{g(D,A)}{H(D)} \tag{5.10}$$

基尼系数：

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

基尼系数其实就是对上的一个近似。在熵的计算公式中，将对数部分按x=1写出一阶泰勒展示，忽略高阶无穷小就得到了基尼系数（不严谨，只是量级上的近似）



分类误差率应该是min（p，1-p）

3.决策树的生成

3.1 ID3算法

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. ■

3.2 C4.5算法

C4.5算法与ID3算法类似，C4.5在生成的过程中用信息增益率来选择最优特征

4. 决策树的剪枝

决策树生成算法对训练数据效果很好，但是会过拟合，需要通过剪枝来提高泛华能力。思路很简单，就是在预测误差（损失）和模型复杂度之间进行权衡。通过最小化损失函数来进行剪枝。

决策树学习的损失函数为：

$$C_\alpha(T)=C(T)+\alpha|T|$$

其中，

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

，α为调节参数，|T|为叶节点的个数，t是树的叶节点，该叶节点有Nt个样本点，其中k类的样本点有Ntk个，$H_t(T)$为叶节点t上的经验熵。

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. ■

5.CART算法

CART即分类回归树。

CART算法假设决策树是二叉树，内部节点特征取值为是和否，这等价于递归的二分每个特征，将输入空间（特征空间）划分为有限个单元，并在这些单元上确定预测的概率分布。CART算法由两部分组成，决策树生成和决策树剪枝。

5.1 CART生成

决策树的生成就是递归生成二叉树的过程。对回归树采用平方误差最小化准则，对分类树采用基尼指数最小化准则进行特征选择。

5.1.1 回归树生成

回归树的输出不是类别而是连续变量。

假设输入空间已经被分为M个单元R1，R2，...，Rm，分别对应输出值cm，于是回归树模型可以表示为：

$$f(x)=\sum_{m=1}^{M}c_mI(x\in R_m)$$

回归树的预测误差为：

$$\sum_{x_i\in R_m}(y_i-f(x_i))^2$$

那么输出值就是使上面误差最小的值，也就是均值：

$$\hat{c}_m=\text{ave}(y_i\mid x_i\in R_m)$$

难点在于怎么划分，一种启发式的方法（其实就是暴力搜索吧）：

遍历所有输入变量，选择第j个变量和它的值s作为切分变量和切分点，将空间分为两个区域：

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

然后计算两个区域的平方误差，求和，极小化这个和，具体的，就是：

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

当j最优化的时候，就可以将切分点最优化：

$$\hat{c}_1=\text{ave}(y_i\mid x_i\in R_1(j,s))\quad 和\quad \hat{c}_2=\text{ave}(y_i\mid x_i\in R_2(j,s))$$

递归调用此过程，这种回归树通常称为最小二乘回归树。

5.1.2分类树的生成

与会归属类似，区别在于使用基尼系数进行特征选择。

设节点的当前数据集为D，对D中每一个特征A，对齐每个值a根据D中样本点是否满足A==a分为两部分，计算基尼指数。对所有基尼指数选择最小的，对应的特征和切分点作为最优特征和最优切分点，生成两个子节点，将对应的两个分区分配过去，然后对两个子节点递归。

5.2 CART剪枝（没太明白）

在上面介绍的损失函数中，当α固定时，一定存在使得损失函数最小的子树，记为复杂度=Tα，α偏大Tα就偏小。设对α递增的序列，对应的最优子树序列为Tn，子树序列第一棵包含第二棵，依次类推。

从T0开始剪枝，对它内部的任意节点t，只有t这一个节点的子树的损失函数是：

$$C_\alpha(t) = C(t) + \alpha$$

以t为根节点的子树的损失函数是：

$$C_\alpha(T_t) = C(T_t) + \alpha|T_t|$$

当α充分小，肯定有

$$C_\alpha(T_t) < C_\alpha(t)$$

这个不等式的意思是复杂模型在复杂度影响力小的情况下损失函数更小。

当α增大到某一点，这个不等式的符号会反过来。

只要

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

，损失函数值就相同，但是t更小啊，所以t更可取，于是把Tt剪枝掉。

为此，对每一个t，计算

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

表示损失函数的减少程度，从T中剪枝掉g(t)最小的Tt，取新的α=g(t)，直到根节点。这样就得到了一个子树序列，对此序列，应用独立的验证数据集交叉验证，选取最优子树，剪枝完毕。

5.2 剪枝算法综述

剪枝算法一般分为预剪枝和后剪枝。

5.2.1 预剪枝

预剪枝就是在决策树生成过程中，做一些判断，不满足判断条件的话就停止决策树的生长。

在一般的决策树中，通常有一些超参数，比如最少分裂样本数，误差下降阈值等，都可以看成预剪枝。

还有一种预剪枝是利用验证集。在树生长的过程中，不断判断当前生长会不会降低验证集的精确度，来决定是否进行生长。

预剪枝容易引起欠拟合，因为很有可能在当前生长节点表现不好，但继续生长后表现又回转。

5.2.2 后剪枝

后剪枝一般包括三种方法。

a REP（错误率剪枝）

思想就是考虑树上的每一个节点，试图对其进行剪枝，并重新决定剪枝后的输出，通过验证集判断错误率有没有降低

b. CCP（代价复杂度剪枝）

思想是最小化树的复杂度，即最少的叶子节点数。通常会把完全生长的决策树，自底向上不断剪枝生成一系列子树，得到子树序列，然后通过交叉验证的方法得到最优子树。

具体的损失函数见上CART剪枝部分。其中的α代表着在最小损失和复杂度之间的权衡。

CART就是使用的CCP剪枝

c. PEP（悲观剪枝）

C4.5采用的就是这种剪枝方法

悲观剪枝不需要使用到验证集。

其思想主要如下：

在每一个叶子节点处都会有一个误判率，因为通常决定把一个节点当做叶子节点的条件是最小分裂样本数。所以肯定会有误判点。设叶子节点的误判率为e

考虑一颗子树，它的误判率是各个叶子节点的误判率的和。这种情况下，如果将一颗子树剪枝为叶子结点的话，它的误判率肯定会上升。

所以悲观剪枝的做法是在误判率上加上一个惩罚因子，通常取0.5，那么现在的情况是：

对一颗子树来说，它的总误判率变成了所有子树的（ei + 0.5）的和，如果把它剪枝为叶子节点，它的误判率就变成了 e’+0.5

因此，剪不剪枝是有一个权衡在里边。