

一、决策树

1.1 简述一下决策树的原理

决策树是一种基本的分类与回归方法，其模型就是用一棵树来表示我们的整个决策过程。这棵树可以是二叉树（比如**CART**只能是二叉树），也可以是多叉树（比如**ID3**、**C4.5**可以是多叉树或二叉树）。

根节点包含整个样本集，每个叶节点都对应一个决策结果（注意，不同的叶节点可能对应同一个决策结果），**每一个内部节点都对应一次决策过程或者说是一次属性测试。从根节点到每个叶节点的路径对应一个判定测试序列。

决策树分为三部分：特征选择，树的生成，树的剪枝：

- 特征选择。特征选择的目的是选取能够对训练集分类的特征。特征选择的关键是准则：信息增益、信息增益比、Gini 指数；
- 决策树的生成。通常是利用信息增益最大、信息增益比最大、Gini 指数最小作为特征选择的准则。从根节点开始，递归的生成决策树。相当于是不断选取局部最优特征，或将训练集分割为基本能够正确分类的子集；
- 决策树的剪枝。决策树的剪枝是为了防止树的过拟合，增强其泛化能力。包括预剪枝和后剪枝。

1.2 决策树如何来划分属性

ID3 使用信息增益作为选择特征的准则；

信息增益：得知特征X的信息而使得类Y的信息的不确定性减少的程度。

信息增益 = 划分前熵 - 划分后熵 $Gain(D,a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$ $Ent(D) = -\sum_{k=1}^{|Y|} p_{klog_2 p_k}$ 信息增益越大，则意味着使用属性a来进行划分所获得的“纯度提升”越大。

ID3 仅仅适用于二分类问题。ID3 仅仅能够处理离散属性。

C4.5 使用信息增益比作为选择特征的准则；

C4.5 克服了 ID3 仅仅能够处理离散属性的问题，以及信息增益偏向选择取值较多特征的问题，使用信息增益比来选择特征。

信息增益比 = 信息增益 / 划分前熵 $Gain_ratio(D, a) = \frac{Gain(D, a)}{IV(a)}$ $IV(a) = -\sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$ C4.5 处理连续特征是先将特征取值排序，以连续两个值中间值作为划分标准。

尝试每一种划分：

1. 计算修正后的信息增益。
2. 选择信息增益最大的分裂点。

CART 使用Gini指数作为选择特征的准则。

CART 生成的树必须是二叉树。CART 的全称是分类与回归树。

- 回归树
使用平方误差最小化准则来选择特征并进行划分。
每一个叶子节点给出的预测值，是划分到该叶子节点的所有样本目标值的均值。

- 分类树使用 **Gini 指数最小化** 准则来选择特征并进行划分。
 假设有 k 个类别，样本点属于第 k 类的概率为 p_k ，则基尼系数为：

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

$$Gini_index(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} Gini(D^v)$$

1.3 信息增益与信息增益比有什么区别

信息增益总是偏向于选择取值较多的属性。信息增益比在此基础上增加了一个罚项，解决了这个问题。

1.4 为什么Gini指数比信息增益和信息增益比要好？

Gini 指数的计算不需要对数运算，更加高效；

Gini 指数更偏向于连续属性，熵更偏向于离散属性。

1.5 如何防止决策树过拟合？

决策树算法很容易过拟合（overfitting），剪枝算法就是用来防止决策树过拟合，提高泛化性能的方法，剪枝分为预剪枝与后剪枝。

预剪枝是指在决策树的生成过程中，对每个节点在划分前先进行评估，若当前的划分不能带来泛化性能的提升，则停止划分，并将当前节点标记为叶节点。

后剪枝是指先从训练集生成一颗完整的决策树，然后自底向上对非叶节点进行考察，若将该节点对应的子树替换为叶节点，能带来泛化性能的提升，则将该子树替换为叶节点。

1.6 如何判断是否带来了泛化性能的提升？

留出法：预留一部分数据作为验证集来进行性能评估。交叉验证

1.7 C4.5决策树如何处理连续数值型属性？

- 先离散 选取一个分界点 v_j ，小于等于 v_j 在左子树，大于 v_j 在右子树。具体步骤为：
- 针对特征的取值进行升序排序
- 取两个特征取值之间的中点作为可能的分裂点。计算信息增益。
- 选择信息增益最大的分裂点作为最佳分裂点。
- 计算最佳分裂点的信息增益率作为特征的信息增益率。根据信息增益率来选择最佳分类特征。
 当离散属性和连续属性并存时，C4.5倾向于选择连续特征。

1.8 决策树如何处理缺失值？

- 如果还未确定用哪个属性来进行分类时 忽略有缺失值的样本；用均值或众数填充；计算信息增益时根据确实样本个数所占比率进行打折。
- 如果确定了属性，在训练阶段发现缺失值 忽略这些样本；众数大法；把属性缺失样本分配给所有子集
- 如果训练完了，面对测试集中的缺失值 众数大法

1.9 如果决策树的属性用完了，但仍未对决策树完成划分怎么办？(欠拟合)

对这些子集进行多数表决，即使用此子集中出现次数最多的类别作为此节点类别，然后将此节点作为叶子节点。

1.10 决策树需要进行归一化处理嘛？

概率模型不需要归一化，因为不关心变量的值，只关心变量的分布和变量之间的条件概率。

1.11 决策树一定是二叉树嘛？

ID3和C4.5就不是二叉树，CART一定是二叉树。

1.12 为什么选择决策树？(说说优缺点)

优点：

1. 可解释性强
2. 不需要归一化，数据处理简单，而且可以同时处理离散和连续数据。

缺点：

1. 容易过拟合
2. 对异常值很敏感
3. 泛化能力太差

二、随机森林

2.1 解释一下什么是集成学习方法

集成学习（ensemble learning）通过构建并组合多个学习器来完成学习任务。

原则：要获得比单一学习器更好的性能，个体学习器应该好而不同。即个体学习器应该具有一定的准确性，不能差于弱学习器，并且具有多样性，即学习器之间有差异。

根据个体学习器的生成方式，目前集成学习分为两大类：

个体学习器之间存在强依赖关系、必须串行生成的序列化方法。代表是 Boosting。

个体学习器之间不存在强依赖关系、可同时生成的并行化方法。代表是 Bagging。

2.2 解释一下什么是 Bagging

Bagging 给出的做法就是对训练集进行采样，产生出若干个不同的子集，再从每个训练子集中训练一个基学习器。

Bagging 是并行式集成学习方法的代表，采样方法是自助采样法（**bootstrap**），用的是有放回的采样。初始训练集中大约有 63.2% 的数据出现在采样集中。

Bagging 在预测输出进行结合时，对于分类问题，采用简单投票法；对于回归问题，采用简单平均法。

Bagging 主要关注降低方差。（**low variance**）

2.3 介绍一下随机森林的原理

Ramdon Forest 在以CART决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入随机属性选择。

1. 先从该节点的特征集中通过bootstrap的方法随机选择 K 个特征的子集

2. 然后从这个特征子集中通过决策树算法选择最优特征进行划分。

2.4 随机森林有什么优缺点？

优点：

1. 只考虑一个属性子集，因此开销小，训练效率高
2. 增加了扰动，集学习器的性能降低。但是基学习器的数量越多，泛化误差越低
3. 引入了随机性后，不容易过拟合
4. 可以处理离散数据和连续数据，而且不需要归一化

缺点：

1. 对于噪声较大的训练集，容易过拟合

2.5 随机森林随机在哪里？

1. 随机森林的随机性体现在每棵树的训练样本是随机的。
2. 随机森林的树种每个节点的分裂属性集合也是随机选择确定的。

2.6 为什么随机森林不容易过拟合？

因为引入随机性后，让每一棵树拟合的细节不同，可以把过拟合的部分会被消除。

2.7 建立了10000棵树的随机森林。如果训练误差为0.00，但是验证错误是34.23。为什么？

这说明过拟合了。为了避免这些情况，我们用交叉验证来调整树的数量。

2.8 如何使用随机森林去弥补特征向量中的缺失值？

1. 对于训练集中的缺失值，使用均值，0等方式进行预填充。
2. 然后使用随机森林分类，更新缺失值。如果分类变量缺失，用众数补；如果连续型变量缺失，使用中位数补。
3. 再次使用随机森林分类更新缺失值。

2.9 随机森林的袋外数据(oob)误差的计算方法？

假设袋外数据总数为O，用这O个袋外数据作为输入，带进之前已经生成的随机森林分类器，分类器会给出这O个数据相应的分类。统计随机森林分类器的分类错误的类目X。\$\$袋外数据误差大小=X/O\$\$

2.10 如何使用随机森林对特征重要性进行评估？

1. 对于随机森林中的每一颗决策树，使用相应的OOB数据来计算它的袋外数据误差，记为errOOB1
2. 随机地对袋外数据OOB所有样本的特征X加入噪声干扰，再次计算它的袋外数据误差，记为errOOB2
3. 假设随机森林中有Ntree棵树，那么 \$\$特征X的重要性=\sum(errOOB2-errOOB1)/Ntree\$\$

2.11 随机森林算法训练时主要需要调整哪些参数？

1. n_estimators 建立子树的数量
2. max_features 允许单个决策树使用特征的最大数量

3. max_depth 决策树的最大深度
4. min_samples_split 内部节点再划分所需最小样本数
5. min_samples_leaf 叶子节点最少样本数
6. max_leaf_nodes 最大叶子节点数
7. min_impurity_split 节点划分最小补纯度

三、Adaboost

3.1 介绍一下Adaboost

1. 给数据中心每一个样本一个权重，权重的起始值是相同的，若有n个训练样本，其权重均为1/n。
2. 训练数据中心的每一个样本，得到第一个分类器。
3. 计算该分类器的错误率，根据错误率计算要给分类器分配的权重。错误率 $\epsilon = \sum_{i \in \text{错分类样本}} W_i$ 分类器权重 $\alpha = \frac{1}{2} \ln \left(\frac{1 - \epsilon}{\epsilon} \right)$
4. 将第一个分类器分错误的样本权重增加，分对的样本权重减小。错误样本权重更新公式： $D_i^{t+1} = \frac{D_i^t e^{\alpha}}{\sum(D_i^t)}$ 正确样本权重更新公式： $D_i^{t+1} = \frac{D_i^t e^{-\alpha}}{\sum(D_i^t)}$
5. 然后再用新的样本权重训练数据，得到新的分类器，到步骤3
6. 直到步骤3中分类器错误率为0或者整体弱分类器错误为0，或者到达迭代次数
7. 将所有弱分类器加权求和，得到分类结果，错误率低得分分类器获得更高的决定系数，从而在对数据进行预测时起关键作用。 $f(x) = \sum \alpha h_i(x)$

四、GBDT

4.1 介绍一下GBDT

GBDT是集成学习Boosting的一种。

Gradient Boosting的主要思想是：每一次建立单个学习器时，是在之前建立的模型的损失函数的梯度下降方向。

GBDT的核心在于：每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得到真实值的累加量。

为了得到残差，GBDT的树是回归树。

算法流程:

1. 初始化弱学习器。 $T(x; \Theta_m)$
2. 对迭代论述 $m=1, 2, \dots, T$:
 1. 计算各个叶子区域损失函数L的负梯度值，将它作为残差的估计 $r_{mi} = -\left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)} \right] f(x) = f_{m-1}(x)$
 2. 对于 r_{mi} 拟合为一棵新的回归树，根据新的回归树得到m轮产生的叶子节点区域
 3. 遍历回归树所有叶子节点区域，在各个区域使损失函数极小化找到残差
 4. 更新强学习器
3. 得到输出的最终模型 $f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$

4.2 GBDT中为什么要用负梯度来代替残差计算？

使用梯度计算代替残差的主要原因是为了将GBDT扩展到更复杂的损失函数中。

4.3 GBDT如何用于分类？


首先必须声明：**GBDT**使用的一定是**CART**回归树。然后如何进行分类？

1. 我们在训练的时候，是针对样本 X 每个可能的类都训练一个分类回归树。把分类问题转化为多个回归问题。
例如：样本有三类，如果样本 x 属于第二类，那么分类结果用向量表示： $[0, 1, 0]$ 。针对样本有三类的情况，实质上在每轮的训练过程中，是同时训练三棵树。第一棵树输入 $(x, 0)$ ，第二棵树输入 $(x, 1)$ ，第三棵树输入 $(x, 0)$ 。这样把一个三分类问题转化为三个回归问题。
2. 假设三棵树对 x 类别的预测分别为 $f_1(x)$, $f_2(x)$, $f_3(x)$ 。最后使用**Softmax**产生概率。 x 属于类别1的概率为： $p_1 = \frac{\exp(f_1(x))}{\sum_{k=1}^3 \exp(f_k(x))}$
3. 最后分别对这些概率计算残差： $y_{11} = 0 - p_1(x)$, $y_{22} = 1 - p_2(x)$, $y_{33} = 0 - p_3(x)$
4. 开始第二轮训练时，第一棵树的输入为 $(x, y_{11}(x))$ ，同理第二第三棵树分别为 $(x, y_{22}(x))$, $(x, y_{33}(x))$
5. 一直迭代 M 轮，每轮构建3棵树，经过 T 轮迭代后，生成 $3 \times T$ 棵树，样本属于某个类别 c 的概率为： $p_c = \frac{\exp(f_c(x))}{\sum_{k=1}^3 \exp(f_k(x))}$

4.4 如何使用GBDT构建特征？

主要思想：**GBDT**每棵树的路径直接作为其它模型的输入特征使用。

GBDT本身不能产生特征，但可以利用GBDT模型学习到的树来构造新特征。

 有一张图

4.5 为什么GBDT不适合使用高维稀疏特征？

1. 特征太多，跑不动
2. 树的分割往往只考虑了少量特征，大部分特征用不到。特征很浪费啊。

4.6 GBDT如何进行正则化？

1. 步长(Learning Rate)
原来的学习器迭代公式为： $f_k(x) = f_{k-1}(x) + h_k(x)$
加上步长正则化后为： $f_k(x) = f_{k-1}(x) + v h_k(x)$
 $0 < v < 1$ 。v小说明需要更多的若学习器迭代次数。通常使用步长和迭代最大次数一起作为算法的拟合效果。
2. 子采样比例(Subsample) 此处采样与随机森林的不同。随机森林是有放回抽样，GBDT是不放回。
选择子采样比例小于1，即选择一部分样本去做GBDT的拟合。可以防止过拟合，减小方差，但是会增大偏差。推荐 $[0.5, 0.8]$ 之间。
3. 对弱学习器(CART回归树)进行正则化剪枝

4.6 GBDT需要调试的参数有哪些？

1. `n_estimators`: 弱学习器的最大迭代次数。(过小欠拟合，过大过拟合)
2. `learning_rate`: 每个弱学习器的权重缩减系数。
3. `subsample`: 不放回抽样
4. `max_features`: 允许单个决策树使用特征的最大数量
5. `max_depth`: 决策树的最大深度
6. `min_samples_split`: 内部节点再划分所需最小样本数
7. `min_samples_leaf`: 叶子节点最少样本数

8. max_leaf_nodes: 最大叶子节点数。(防止过拟合, 因为默认为None)
9. min_impurity_split: 节点划分最小不纯度

4.7 GBDT算法的优缺点有哪些?

优点:

1. 可以灵活处理各种类型的数据, 包括连续值和离散值
2. 可以使用高级的损失函数, Huber损失函数, Quantile损失函数
3. 在相对少的调参时间下, 预测的准确率可以比较高
- 缺点:
4. 弱学习器之间存在强依赖关系, 难以并行训练数据。

五、XGBoost

5.1 为什么Xgboost要用泰勒展开, 优势在哪里?

Xgboost使用了一阶和二阶偏导, 二阶导数有利于梯度下降的更快更准。

使用泰勒展开取得函数做自变量的二阶导数形式, 可以在不选定损失函数具体形式的情况下, 仅仅依靠输入数据的值来进行叶子分裂优化计算, 本质上也就把损失函数的选取和模型算法优化分开了。

5.2 Xgboost如何寻找最优特征?

Xgboost在训练的过程中给出各个特征的增益评分, 最大增益的特征会被选出来作为分裂依据, 从而记忆了每个特征对在模型训练时的重要性, 从根到叶子中间节点涉及某特征的次数作为该特征重要性排序。

5.3 Xgboost采样是有放回还是无放回呢?

样本是不放回的, 因此每轮计算样本不重复。

相比于GBDT, Xgboost同时支持子采样和列采样, 即可以随机采样特征, 防止过拟合。

5.4 Xgboost和GBDT的区别。

1. GBDT的弱学习器一定是CART树。XGboost还支持带L1和L2正则化的逻辑回归和线性回归。
2. GBDT优化时, 只使用一阶导数信息。XGBoost对代价函数使用了二阶泰勒展开。
3. XGBoost在代价函数中加入正则化, 用于控制模型的复杂度, 损失函数如下:
$$\text{Obj} = \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$
$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$
4. XGBoost支持列抽样, 对特征进行随机抽样
5. XGBoost工具支持并行