LightGBM

1. 综述

微软在 2017 年的时候在 nips 上发表了一篇论文，并开源了相关工具，也就是出名的 LGB 。下面有三个链接非常的重要，是 LGB 重要的参考工具。

lgb官方文档
lgb–github
lgb论文

lgb 模型是经典的 GBDT 模型的高效实现方式，lgb 模型可以针对机器学习中的 classfication 问题，regression 问题和 ranking 问题进行建模。

首先我们从名字上对相关模型做一些释义。

GBDT(Gradient Boosting Decision Tree)即梯度提升决策树。它还有另外一个名字 MART(Multiple Additive Regression Tree)。GBDT 有时候又会叫做 GBRT(Gradient Boosting Regression Tree)。更准确点说 GBDT 应该是 MART 的一种实现方式。一般我们也将这三者等价，他们都是迭代的决策树算法，通过将多棵决策树组合，所有决策树的结果加起来得到最终的答案。（这里的决策树一般指 CART 决策树）

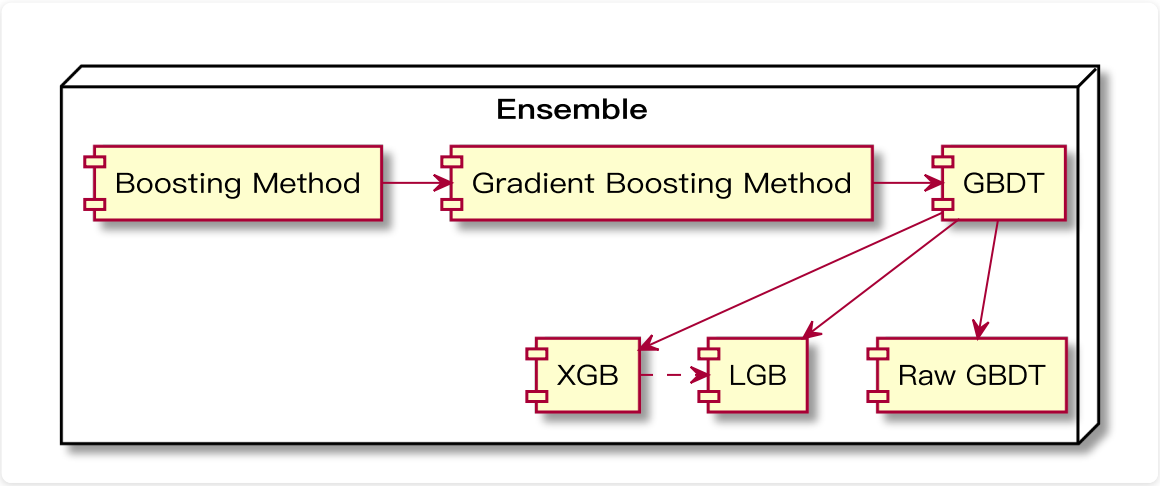
那是不是一定要用 Decision Tree 作为基模型呢？答案是否定的。如果我们将 Decision Tree 推广到一般的基模型，就能得到一类通用的提升方法，这类方法称为 GBM 方法（Gradient Boosting Method）。

那是不是一定要用梯度来解决 Boosting 模型(前向加法模型)呢？答案也是否定的，解决 Boosting 模型的方案有很多，用 Gradient 来解决是较好和较高效的一种。

最后我们再回到 GBDT，GBDT 的实现方式有很多，有比较原始的 GBDT 的实现，有在 GBDT 上进行改进，得到的 XGB 的实现。XGB 的实现其实和 GBDT 已经有很大的不同了。这个不同体现在树的分裂方式和叶子结点值的确定方式。核心思想是对 GBDT 待拟合的损失函数做二阶泰勒展开，并巧妙引入树的正则项。使得可以对二阶泰勒展开的公式进行化简和解析求解。从而推导出了新的树分裂方式和叶子节点值确定方式。

LGB 则是在 XGB 对 GBDT 的公式改进基础上，做的进一步的优化。

所以我们有如下的关系：



---

2. 前置知识

2.1 CART 决策树

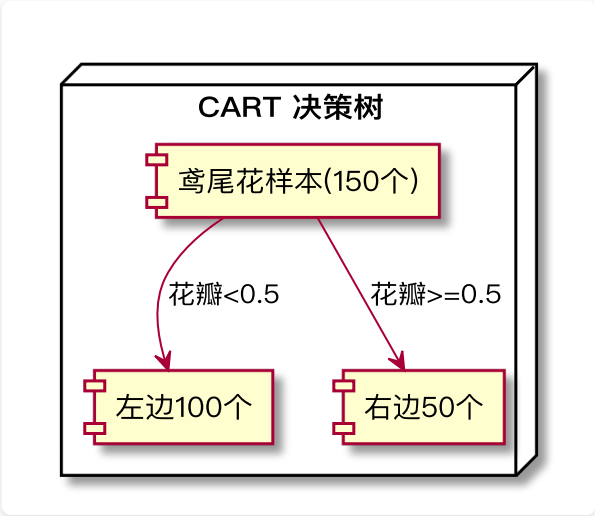
决策树的实现方式有很多，比如经典的 ID 3.0，C4.5 商业版的C5.0。业界用的最多的是另一个虽然简单但是却能衍生出很多高级模型的决策树 CART 决策树。CART 决策时又名“分类回归树”，是一种简单的二叉树，即可以做分类，又可以做回归。

无论是何种决策树，他们的本质思想都可以用一句话来概括。

> *对训练集进行划分，使得划分后的集合的纯度变得"更纯"。*

从这句话能看到，对于决策树有三件事很重要：

1. 如何定义集合的纯度。（划分前和划分后）
2. 如何对集合进行划分。（选择哪个特征和阈值）
3. 如何确定叶子节点的值。（决定了预测结果）



如何定义纯度：

回归问题：标签连续，使用方差定义纯度。
分类问题：标签离散，使用熵和GINI指数定义。

注意：划分前后，集合由一个变为了两个，所以需要对纯度进行加权求和。

如何进行划分：

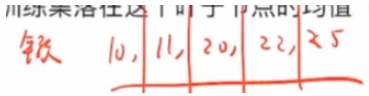
遍历特征，遍历样本对应的特征值。分别计算纯度和，选择纯度提升最大的那个。

叶子节点值的确定：

当我们划分到叶子节点后，这个叶子节点对应的**预测值**，就是训练集落在这个叶子节点的均值（连续标签为均值，类别标签为类别占比）

计算量统计：

每分裂一次的计算量：



$$cost_{time} = feature_{num} \times sample_{num} \times point_{num} \tag{1}$$

2.2 GBDT 模型

回顾前向加法模型：

$$f(x) = \sum_{i=1}^{M} \beta_i h(x; \theta_i)$$

x 表示样本，$h(x; \theta_i)$ 表示第 i 个基模型。所以整个模型是 M 个模型的加权求和。所以这个模型称为前向加法模型。

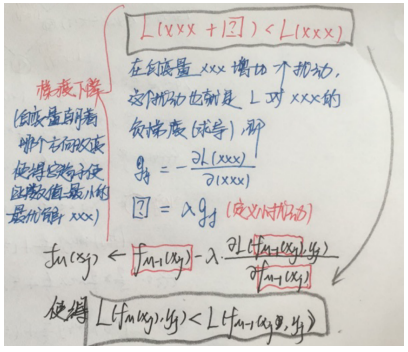
假设我们要解决的问题的损失函数是 L，L 可能是 MSE（解决回归问题），也可能是交叉熵和指数损失（解决分类问题），甚至可以是你自定义的损失函数。

我们期望学习到的模型 f 为使得损失函数 L 尽量小的 f。

小扰动怎么定义，就是梯度下降，令函数对自变量进行求导

$$\min \sum_{j=1}^{N} L(f(x_j), y_j) = \min \sum_{j=1}^{N} L(\sum_{i=1}^{M} \beta_i h(x_j; \theta_i), y_j) \tag{2}$$

$L(xxx+?) < L(xxx)$



$\{(x_j, y_j)\}_{j=1}^{N}$ 为训练样本。 梯度下降的原理相同，已知当前的f(x)，要确定使f(x)减小最快的一个x，就利用梯度使得x逼近最小的x
参考：https://zhuanlan.zhihu.com/p/43452377

如何进行训练？毫无疑问，只能使用贪心的方法进行训练，否则这是一个 NP-hard 问题。即假设已知 $\sum_{i=1}^{M-1} \beta_i h(x; \theta_i)$，求 $\beta_M h(x; \theta_M)$ 使得 (2) 尽量的小。

朴素的想法就是针对每个样本 x_j，找到 $\beta_M h(x_j;\theta_M)$ ，使得 $L(f_{M-1}(x_j)+\beta_M h(x_j;\theta_M),y_j)$，尽量小。于是我们可以通过梯度下降法。让 $\beta_M h(x_j;\theta_M),y_j)$ 去拟合负梯度 g_j，这里负梯度 g_j 为：

$$g_j=-\frac{\partial L(f_{M-1}(x_j),y_j)}{\partial f_{M-1}(x_j)}$$

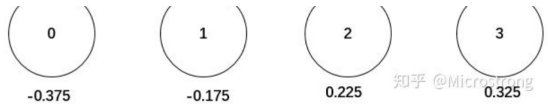
对于每一个样本都有一个 g_j，所以有新的样本对 $\{(x_j,g_j)\}$ 拟合的方式就转换为一个回归问题：

$$\theta_M,\beta_M=\arg\min_{\theta,\beta}\sum_{j=1}^{N}||g_j-\beta h(x_j;\theta)||^2$$

$$r_{m,i}=-\left[\frac{\partial L(y_i,F(x_i))}{\partial F(x)}\right]_{F(x)=F_{m-1}(x)}$$

现将残差的计算结果列表如下：

编号	真实值	$F_0(x)$	残差
0	1.1	1.475	-0.375
1	1.3	1.475	-0.175
2	1.7	1.475	0.225
3	1.8	1.475	0.325



拟合出了负梯度后，还需要去确定最终的步长。

$$\rho_m=\arg\min_{\rho}\sum_{j=1}^{N}L(f_{M-1}(x_j)+\rho h(x_j),y_j)$$

所以最终的模型为：

https://zhuanlan.zhihu.com/p/81016622?from_voters_page=true
深入理解GBDT回归算法

$$f(x)=f_{M-1}+\rho_M\beta_M h(x;\theta_M)$$

此时可更新强学习器，需要用到参数学习率：learning_rate=0.1，用 lr 表示。

$$F_1(x)=F_0(x)+lr*\sum_{j=1}^{4}c_{1,j}I(x\in R_{1,j})$$

这就是出名的 GBM 方法。 GBM 方法是一类通用的用梯度方式来学习前向加法模型的方法。这里我们并未要求基学习器 h 一定是决策树。

当我们把 h 确定为 cart 决策树时，我们的方法就变为了出名的 GBDT 方法。

$\theta_M,\beta_M=\arg\min_{\theta,\beta}\sum_{j=1}^{N}||g_j-\beta h(x_j;\theta)||^2$GBDT 方法相比于通用的 GBM 方法会略有一些变化。这种变化体现在ρ_M 的确定。当用决策树去回归负梯度 g_j 的时候，实际上就形成了关于特征空间的一个划分（每个叶子结点就是划分出来的一个区域）。那么我们就可以在每一个叶子区域上去确定最佳的 ρ_{Mj}，这里 M 表示第 M 个决策树，j 表示这个决策树对应的一系列的叶子结点。于是我们有了如下的 GBDT 算法：

1. 假设已知 $f_{M-1}(x)=\sum_{i=1}^{M-1}h(x;\theta_i)$，求 $h(x;\theta_M)$。

2. 计算每一个样本关于损失函数的梯度，$g_j = -\frac{\partial L(f_{M-1}(x_j), y_j)}{\partial f_{M-1}(x_j)}$，得到 $\{(x_j, g_j)\}_{j=1}^{N}$。
3. 使用 cart 决策树，回归 $\{(x_j, g_j)\}_{j=1}^{N}$ ，得到特征空间的划分（也就是叶子结点）。记为 $\{R_{Mj}\}_1^J$，注意这个时候叶子节点的值没有确定！
4. 对每一个叶子节点，遍历获得这个叶子节点值的最佳预测。

这里其实和上面初始化弱学习器是一样的，对平方损失函数求导，令导数等于零，化简之后得到每个叶子节点的参数 c，其实就是标签值的均值。这个地方的标签值不是原始的 y，而是本轮要拟合的标残差 $y - f_0(x)$。

$$c_{Mj} = \arg\min_{c} \sum_{x_i \in R_{Mj}} L(f_{M-1}(x_i) + c)$$

c) 对于 J_m 个叶子节点区域 $j = 1, 2, \ldots, J_m$，计算出最佳拟合值：

$$c_{m,j} = \arg\min_{c} \sum_{x_i \in R_{m,j}} L\left(y_i, F_{m-1}\left(x_i\right) + c\right)$$

https://zhuanlan.zhihu.com/p/81016622?from_voters_page=true 例子

5. 这个时候，我们有了叶子节点划分 R_{Mj}，和每个叶子节点对应的值 c_{Mj}，这时候就得到了一颗新树 $h_M(x;\theta_M)$。

对于 GBDT，大家把决策树的三个核心问题再思考一遍：

1. 如何定义集合的纯度？（划分前和划分后）
2. 如何对集合进行划分？（选择哪个特征和阈值）
3. 如何确定叶子节点的值？（决定了预测结果）

2.3 从 GBDT 到 XGB

$$J(f_t) = \sum_{i=1}^{n} L\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + C$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

XGB 的最底层模型仍然是前向加法模型，但不同点在于不是简单通过梯度下降来进行优化，而是巧妙的引入树的正则项和二阶泰勒展开，使得 GBDT 在单个树的分裂和叶子节点值确定变得更加的简单。虽然公式变简单，但是它在某种程度上反而更加精确了。

在这里我们就不再推导，详细推导可见 深度之眼 的XGB的推导视频。

编号	年龄(岁)	体重(kg)	标签值 $\frac{g_j}{h_j}$
0	5	20	-0.375
1	7	30	-0.175
2	21	70	0.225
3	30	60	0.325

GBDT 需要知道每个样本对应的梯度 $\{(x_j, g_j)\}_{j=1}^{N}$，XGB 则需要知道每个样本对应的梯度和 Hess 值(2 阶梯度) $\{(x_j, g_j, h_j)\}_{j=1}^{N}$。GBDT 树的分裂就是通过 CART 决策树回归 g_i 的方式得到树的分裂，叶子节点值的确定是通过遍历（也就是 2.2 节的 4）。XGB 的树的分裂和叶子节点的值的确定均和 (g_j, h_j) 有关，公式如下：$c_{Mj} = \arg\min_{c} \sum_{x_i \in R_{Mj}} L(f_{M-1}(x_i) + c)$

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$$

$$W_j = -\frac{G_j}{H_j+\lambda}$$

γ 和 λ 是衡量树复杂度的两个超参数，分别调整叶子节点数量和叶子节点值平方和的权重。

G_L:表示所有被划分在左边的样本的 g_j 之和。
H_L:表示所有被划分在左边的样本的 H_j 之和。

W_j:表示叶子节点对应的值。
G_j:表示所有被划分在叶子节点 j 这个区域的样本的 g_j 之和。
H_j:表示所有被划分在叶子节点 j 这个区域的样本的 H_j 之和。

通过 $Gain$ ，我们就能知道当下这个划分的纯度是否有“提高”。
通过 W_j ，我们就能计算出叶子节点对应的值。

对于 XGB，大家把决策树的三个核心问题再思考一遍：

1. 如何定义集合的纯度？（划分前和划分后）
2. 如何对集合进行划分？（选择哪个特征和阈值）
3. 如何确定叶子节点的值？（决定了预测结果）

---

3. LGB 模型

LGB 模型是在前面 XGB 的推导下，进行了进一步的优化。这些优化一方面是要减少计算量，另一方面可能能起到防止过拟合的作用（因为原始数据存在噪声，一些粗糙处理可能反而能增加模型的泛化能力）。

从公式 (1) 中，能看到树模型的计算量主要体现在三个方面：

1. 样本数量。（e.g h_i 的计算量和样本数量息息相关）
2. 特征数量。（e.g 树分裂时候需要遍历特征）
3. 候选点数量。（e.g 树分裂时候需要遍历该特征下的候选点）

所以 LGB 的核心技术就是从这三个维度出发，尽量减少每一个维度的计算量。

对应的是那个核心技术就是：

- GOSS（梯度单边采样）：减少样本数量。
- EFB（特征绑定技术）：减少特征数量。
- Hist（直方图算法）：减少候选点数量。

Feature binding

3.1 GOSS

梯度单边采样的核心目的是为了减少样本的数量。

对于一个样本 x_i，如果它的 g_i 比较小，说明这个样本已经拟合的比较好，所以我们可以忽略掉 g_i 比较小的样本值。保留 g_i 比较大的样本值。但是简单的过滤训练样本会导致数据分布的变化，这会严重的影响模型的准确性，于是提出了 GOSS 算法。

假设 a = 20%，b = 30%

1. 计算出所有样本的 g_i，取出最大的前 a。
2. 对剩下的 1–a，随机采样 b 。
3. 在计算 gain 时候，将随机采样的这些样本的 g_i 和 h_i 都乘以 $\frac{1-a}{b}$

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$$

这里着重说一下第三步，因为如果只是简单随机采样，那么在分裂节点的时候，必然会影响 G_i 和 H_i 的值。所以需要保证采样过滤后的 G_i 和原始的 G_i 值要大小大概相等。

这里原论文有一个详细的数学证明，有兴趣同学可以查看一下。

3.2 EFB

特征绑定的核心目的是减少样本的数量，在原论文中 EFB 包含了直方图算法，但是我把它单独拆出来，可以更方便大家理解。所以这里我们只讨论特征的绑定。

特征的绑定的核心思想是认为对于大量的稀疏特征，他们可能是互斥的。比如对于 onehot 编码的特征，只能有一个是 1，其他必然都是 0，这就是一个互斥的特征。所以 EFB 就是想把互斥的特征找出来，把互斥的特征拼成一个特征，减少特征的数量。注意，并不是要求特征完全互斥才绑定在一起，部分互斥也是可以忍受的。

所以需要解决两个问题：

- 需要绑定哪些特征?
- 如何绑定这些特征?

首先完美的找到所有互斥特征是一个 np-hard 问题。所以论文使用了一种贪心的方法来解决这个特征的绑定问题。

创建一个图 (V,E)，V 是顶点，表示特征。E 是边，表示两个顶点是否冲突。如果冲突（e.g 同时非 0 ），则将两个顶点相连。边的权重表示他们冲突的次数。则时候，就转换为了图着色问题，最后的着色结果，相同颜色的顶点（特征）就是一个特征的绑定。

> *给定一个无向图G = (V,E)，其中V为顶点集合，E为边集合，图染色/图着色问题（graph coloring problem, GCP）是将每个顶点涂上颜色，使得每个相邻的顶点着不同的颜色，求出最少使用的颜色数K。*

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow$ G.sortByDegree()
bundles $\leftarrow$ {}, bundlesConflict $\leftarrow$ {}
for i ***in*** *searchOrder* **do**
    needNew $\leftarrow$ True
    for $j = 1$ ***to*** *len(bundles)* **do**
        cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
        if *cnt* + *bundlesConflict[i]* $\leq K$ **then**
            bundles[j].add(F[i]), needNew $\leftarrow$ False
            break
    if *needNew* **then**
        Add $F[i]$ as a new bundle to *bundles*
Output: *bundles*

接下来第二个问题是如何把这些特征绑定在一起，这里会涉及到直方图算法，但直方图算法在下一部分进行讲解，所以让我们来简化的理解一下。

假设现在有特征 A 和 特征 B ，现在已经确定他们两个要绑定在一起。那么当我们把他们绑定为一个特征后，我们需要构建成一个新的特征，且这个特征要能参与树的分裂。假设特征 A 的取值范围在 [0,5] 之间，特征 B 的取值范围在 [0,2] 之间。 那么特征 A 和 B 和在一起时，就有了交集，无法简单通过一个阈值将他们区分开来，于是可以给特征 A 加上一个偏移，比如 2，这样特征 A 就都在 [2,5] 之间，这样特征 A 和 B 就能区分开了。如果特征 A 和特征 B 是不同的类别特征，且类别特征的取值不同，就不会发生这种情况。

Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
for f ***in*** F **do**
    totalBin $+=$ f.numBin
    binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
    newBin[i] $\leftarrow 0$
    for $j = 1$ ***to*** $len(F)$ **do**
        if $F[j].bin[i] \neq 0$ **then**
            newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]

Output: $newBin$, $binRanges$

3.3 Hist

直方图算法的核心目的是减少候选点的数量。

特征可以分为两大类：

- 数值特征
- 类别特征

这两类我们会分别进行讨论：

数值特征

我们会有 $\{(x_{ei}, g_{ei}, h_{ei})\}_{i=1}^{N}$ 这样的样本对（e 表示第 e 个特征），理论上我们需要对每一个 x_{ei} 的取值进行遍历，然后按照他划分， 然后分别计算划分后的 G_{eL}，H_{eL} G_{eR}，H_{eR} 来确定 $Gain$

的值。很明显，即是样本数量进行了减少，这个特征值的遍历和划分，仍然是很恐怖的计算量。于是有了直方图算法来解决这个问题。

直方图算法的步骤如下：

1. 选定一个特征 e，将 $\{(x_{ei}, g_{ei}, h_{ei})\}_{i=1}^{N}$ 按照 x_{ei} 进行排序。
2. 将 x_{ei} 划分成 256 个 bins。
3. 计算每个 bins 中的 $\{(N_{ei}, G_{ei}, H_{ei})\}_{i=1}^{256}$
4. bins 的每个区间就是我们的划分候选点。

注意，上面的直方图是针对一个具体的特征 e 的。现在我们的候选点已经缩小到了 256 。远远小于原始的候选点数量。

当我们对特征 e 从候选点中选择某个阈值进行划分之后，接下来，我们还需要构建划分之后关于特征 e'，左右子集合的直方图。当我们计算出 e' 的左子集合的直方图后，右直方图可以直接通过特征 e 划分之前的直方图相减得到（这个就是直方图差加速算法）。

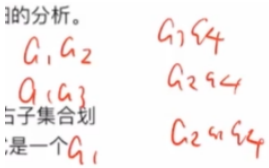
类别特征

类别特征相比于数值特征就会很简单，只需要将这个类别特征中的每一个类别值对应为一个 bins 即可。一般在 lgb 的实现中，会过滤掉那些样本数很小的类别。

最后我们再谈一下候选点遍历的问题。 虽然从前面的直方图算法能看到，数值特征直方图生成过程复杂，类别特征简单。但是在对候选点进行遍历划分的时候，类别特征相比数值特征就麻烦了很多。数值特征只需要用一个阈值进行划分即可，但类别特征则不行。下面我们来进行详细的分析。

类别特征的遍历：

假设有一个类别特征 A，有 a_1, a_2, a_3, a_4， 4 种类别。当我们使用 cart 决策树进行左右子集合划分的时候，一共具有 $2^4 - 1$ 种划分方式。所以对于类别特征来说，找到最优的划分方式是一个 np-hard 问题。



lgb 则使用贪心的方式来解决这个问题，期望能直接对类别特征进行划分，而不是对类别特征进行编码加阈值的方式来处理。

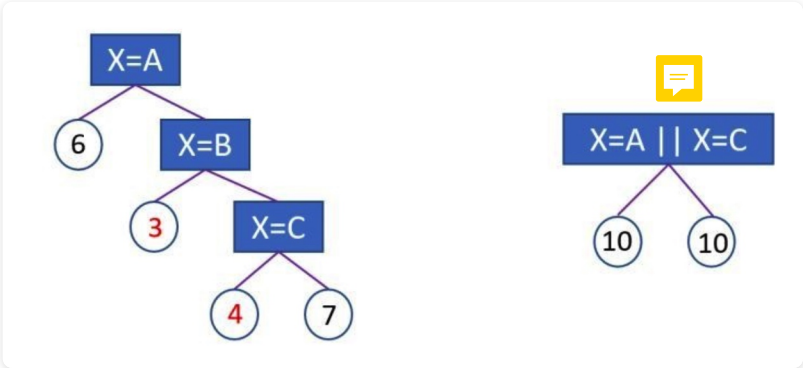
算法大致流程如下：

1. 选定类别特征 e，统计该类别特征下，每一个类别对应的样本数量。过滤掉出现次数少的类别。形成直方图。
2. 计算每个 bins 的“值”

$$\frac{G_j}{H_j + \lambda_{cat}}$$

按照这个值从小到大排序。
3. 分别从左往右和从右往左进行搜索，计算 $Gain$ 值。
4. 找到最优的划分点。

最后是关于这种类别特征划分方法比 one-hot 编码的划分方法更好的原因。



4. 关于其他的基模型

除了 CART 决策树以外。还有一类效果不错基学习器 DART，他是在 CART 决策树的基础上引入了 dropout 的思想。