GBDT二分类

GBDT无论是做分类任务还是回归任务都是使用**CART回归树**作为基分类器。

二分类任务使用与逻辑回归相同的对数似然损失作为损失函数，对强分类器进行偏导（强分类器是多个弱分类器的累加和）。

逻辑回归的对数损失函数

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

函数 $h_\theta(x)$ 的值有特殊的含义，它表示结果取 1 的概率，因此对于输入 x 分类结果为类别 1 和类别 0 的概率分别为：

$$P(Y=1|x;\theta) = h_\theta(x)$$

$$P(Y=0|x;\theta) = 1 - h_\theta(x)$$

下面我们根据上式，推导出逻辑回归的对数损失函数 $L(\theta)$ 。上式综合起来可以写成：

$$P(Y=y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

然后取似然函数为：

$$l(\theta) = \prod_{i=1}^{N} P(Y=y_i|x_i;\theta) = \prod_{i=1}^{N}(h_\theta(x_i))^{y_i}(1-h_\theta(x_i))^{1-y_i}$$

因为 $l(\theta)$ 和 $logl(\theta)$ 在同一 θ 处取得极值，因此我们接着取对数似然函数为：

$$L(\theta) = \sum_{i=1}^{N}[y_i log h_\theta(x_i) + (1-y_i)log(1-h_\theta(x_i))]$$

最大似然估计就是求使 $L(\theta)$ 取最大值时的 θ 。这里对 $L(\theta)$ 取相反数，可以使用梯度下降法求解，求得的 θ 就是要求的最佳参数：

$$J(\theta)=-\frac{1}{N}L(\theta)=-\frac{1}{N}\sum_{i=1}^{N}[y_i log h_\theta(x_i)+(1-y_i)log(1-h_\theta(x_i))]$$

GBDT二分类原理

逻辑回归单个样本 (x_i, y_i) 的损失函数可以表达为：

$$L(\theta)=-y_i\log\hat{y}_i-(1-y_i)\log(1-\hat{y}_i)$$

其中，$\hat{y}_i=h_\theta(x)$ 是逻辑回归预测的结果。假设GBDT第 M 步迭代之后当前学习器为 $F(x)=\sum_{m=0}^{M}h_m(x)$，将 $\hat{y}_i$ 替换为 $F(x)$带入上式之后，可将损失函数写为：

$$L(y_i,F(x_i))=y_i\log\left(1+e^{-F(x_i)}\right)+(1-y_i)\left[F(x_i)+\log\left(1+e^{-F(x_i)}\right)\right]$$

其中，第 m 棵树对应的响应值为（损失函数的负梯度，即伪残差)：

$$r_{m,i}=-\left|\frac{\partial L(y_i,F(x_i))}{\partial F(x_i)}\right|_{F(x)=F_{m-1}(x)}=y_i-\frac{1}{1+e^{-F(x_i)}}=y_i-\hat{y}_i$$

对于生成的决策树，计算各个叶子节点的最佳残差拟合值为：

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

由于上式没有闭式解（closed form solution），我们一般使用近似值代替：

$$c_{m,j}=\frac{\sum_{x_i\in R_{m,j}}r_{m,i}}{\sum_{x_i\in R_{m,j}}(y_i-r_{m,i})(1-y_i+r_{m,i})}$$

GBDT二分类算法训练过程

1. 初始化第一个弱分类器F_0(x)：

$$F_0(x)=log\frac{P(Y=1|x)}{1-P(Y=1|x)}$$

其中，$P(Y=1|x)$ 是训练样本中 $y=1$ 的比例，利用先验信息来初始化学习器。

2. **对于建立 M 棵分类回归树 $m=1,2,\ldots,M$**：

- 对 $i=1,2,\ldots,N$，计算第 m 棵树对应的响应值（损失函数的负梯度，即伪残差）：

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

- 对于$i=1,2,\ldots,N$，利用CART回归树拟合数据 $(x_i,r_{m,i})$，得到第 m 棵回归树，其对应的叶子节点区域为 $R_{m,j}$，其中 $j=1,2,\ldots,J_m$，且 J_m 为第m棵回归树叶子节点的个数。
- 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：

$$c_{m,j}=\frac{\sum_{x_i\in R_{m,j}}r_{m,i}}{\sum_{x_i\in R_{m,j}}(y_i-r_{m,i})(1-y_i+r_{m,i})}$$

- 更新强学习器 $F_m(x)$：

$$F_m(x)=F_{m-1}(x)+\sum_{j=1}^{J_m}c_{m,j}I\left(x\in R_{m,j}\right)$$

3. **得到最终的强学习器 $F_M(x)$ 的表达式**：

$$F_M(x)=F_0(x)+\sum_{m=1}^{M}\sum_{j=1}^{J_m}c_{m,j}I\left(x\in R_{m,j}\right)$$

从以上过程中可知，除了由损失函数引起的负梯度计算和叶子节点的最佳残差拟合值的计算不同，二元GBDT分类和GBDT回归算法过程基本相似。那二元GBDT是如何做分类呢?

将逻辑回归的公式进行整理，我们可以得到 $\log\frac{p}{1-p}=\theta^Tx$，其中$p=P(Y=1|x)$，也就是将给定输入 x 预测为正样本的概率。逻辑回归用一个线性模型去拟合 $Y=1|x$ 这个事件的对数几率（odds）$\log\frac{p}{1-p}$。二元GBDT分类算法和逻辑回归思想一样，用一系列的梯度提升树去拟合这个对数几率，其分类模型可以表达为：

$$P(Y=1|x)=\frac{1}{1+e^{-F_M(x)}}$$

二分类算法实例

（1）数据集介绍

训练集如下表所示，一组数据的特征有年龄和体重，把身高大于1.5米作为分类边界，身高大于1.5米的令标签为1，身高小于等于1.5米的令标签为0，共有4组数据。

编号	年龄(岁)	体重(kg)	身高>1.5(m)(标签)
0	5	20	0
1	7	30	0
2	21	70	1
3	30	60	1

测试数据如下表所示，只有一组数据，年龄为25、体重为65，我们用在训练集上训练好的GBDT模型预测该组数据的身高是否大于1.5米?

编号	年龄(岁)	体重(kg)	身高>1.5(m)(标签)
0	25	65	?

（2）模型训练阶段

参数设置:

- 学习率：learning_rate = 0.1
- 迭代次数：n_trees = 5
- 树的深度：max_depth = 3

1）初始化弱学习器:

$$F_0(x) = log\frac{P(Y=1|x)}{1-P(Y=1|x)} = log\frac{2}{2} = 0$$

2）对于建立M棵分类回归树$m = 1, 2, \ldots, M$:

由于我们设置了迭代次数：n_trees=5，这就是设置了M=5。

首先计算负梯度，根据上文损失函数为对数损失时，负梯度（即伪残差、近似残差）为:

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

我们知道梯度提升类算法，其关键是利用损失函数的负梯度的值作为回归问题提升树算法中的残差的近似值，拟合一个回归树。这里，为了称呼方便，我们把负梯度叫做残差。

现将残差的计算结果列表如下:

编号	真实值	$F_0(x)$	残差
0	0	0	-0.5
1	0	0	-0.5
2	1	0	0.5
3	1	0	0.5

此时将残差作为样本的标签来训练弱学习器 $F_1(x)$，即下表数据：

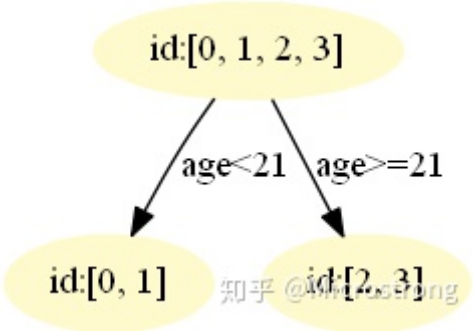
编号	年龄(岁)	体重(kg)	身高>1.5(m)(标签)
0	5	20	-0.5
1	7	30	-0.5
2	21	70	0.5
3	30	60	0.5

接着寻找回归树的最佳划分节点，遍历每个特征的每个可能取值。从年龄特征值为5开始，到体重特征为70结束，分别计算分裂后两组数据的平方损失（Square Error），SE_l 为左节点的平方损失，SE_r 为右节点的平方损失，找到使平方损失和 $SE_{sum} = SE_l + SE_r$ 最小的那个划分节点，即为最佳划分节点。

例如：以年龄7为划分节点，将小于7的样本划分为到左节点，大于等于7的样本划分为右节点。左节点包括 x_0，右节点包括样本 x_1, x_2, x_3，$SE_l = 0$，$SE_r = 0.667$，$SE_{sum} = 0.667$，所有可能的划分情况如下表所示：

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄5	/	0，1，2，3	0.000	1.000	1.000
年龄7	0	1，2，3	0.000	0.667	0.667
年龄21	0，1	2，3	0.000	0.000	**0.000**
年龄30	0，1，2	3	0.667	0.000	0.667
体重20	/	0，1，2，3	0.000	1.000	1.000
体重30	0	1，2，3	0.000	0.667	0.667
体重60	0，1	2，3	0.000	0.000	**0.000**
体重70	0，1，3	2	0.667	0.000	0.667

以上划分点的总平方损失最小为**0.000**，有两个划分点：年龄21和体重60，所以随机选一个作为划分点，这里我们选**年龄21**。现在我们的第一棵树长这个样子：

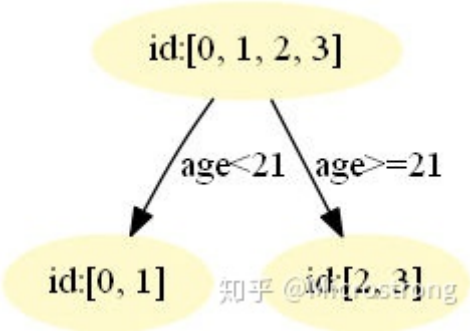


我们设置的参数中树的深度max_depth=3，现在树的深度只有2，需要再进行一次划分，这次划分要对左右两个节点分别进行划分，但是我们在生成树的时候，设置了三个树继续生长的条件：

- **深度没有到达最大。树的深度设置为3，意思是需要生长成3层。**
- **点样本数 >= min_samples_split**

- ***此节点上的样本的标签值不一样（如果值一样说明已经划分得很好了，不需要再分）（本程序满足这个条件，因此树只有2层）***

最终我们的第一棵回归树长下面这个样子：



此时我们的树满足了设置，还需要做一件事情，给这棵树的每个叶子节点分别赋一个参数c，来拟合残差。

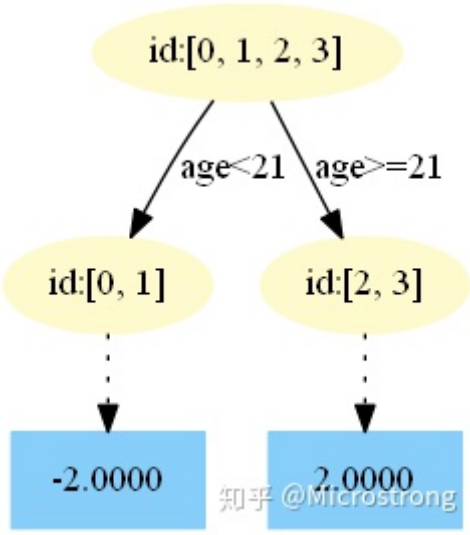
$$c_{1,j} = \frac{\sum_{x_i \in R_{1,j}} r_{1,i}}{\sum_{x_i \in R_{1,j}} (y_i - r_{1,i})(1 - y_i + r_{1,i})}$$

根据上述划分结果，为了方便表示，规定从左到右为第1,2个叶子结点，其计算值过程如下：

$$(x_0, x_1 \in R_{1,1}), \qquad c_{1,1} = -2.0$$

$$(x_2, x_3 \in R_{1,2}), \qquad c_{1,2} = 2.0$$

此时的第一棵树长下面这个样子：



接着更新强学习器，需要用到学习率：learning_rate=0.1，用lr表示。更新公式为：

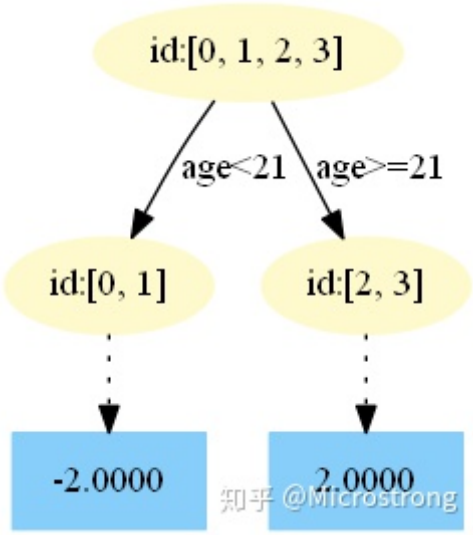
$$F_1(x) = F_0(x) + lr * \sum_{j=1}^{2} c_{1,j} I\left(x \in R_{1,j}\right)$$

为什么要用学习率呢？这是**Shrinkage**的思想，如果每次都全部加上拟合值 c ，即学习率为1，很容易一步学到位导致GBDT过拟合。

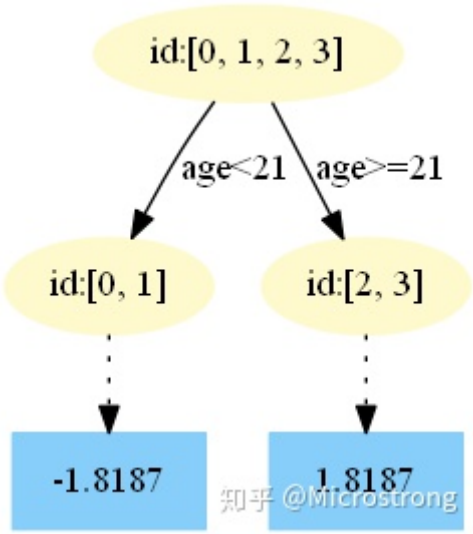
重复此步骤，直到 $m > 5$ 结束，最后生成5棵树。

下面将展示每棵树最终的结构，这些图都是我GitHub上的代码生成的，感兴趣的同学可以去运行一下代码。https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/tree/master/Ensemble%20Learning/GBDT_GradientBoostingBinaryClassifier

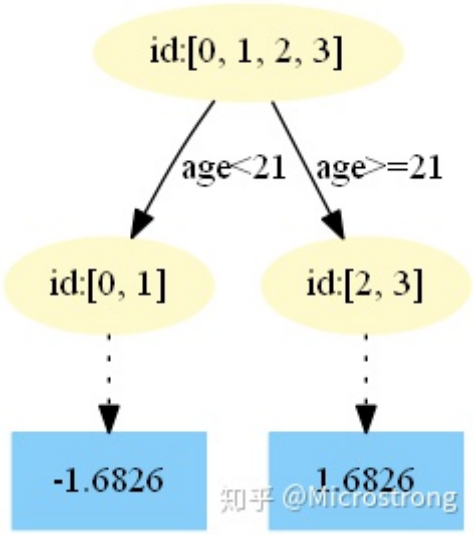
第一棵树：



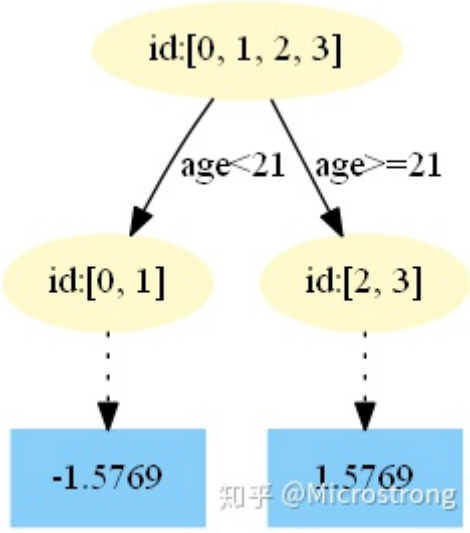
第二棵树：



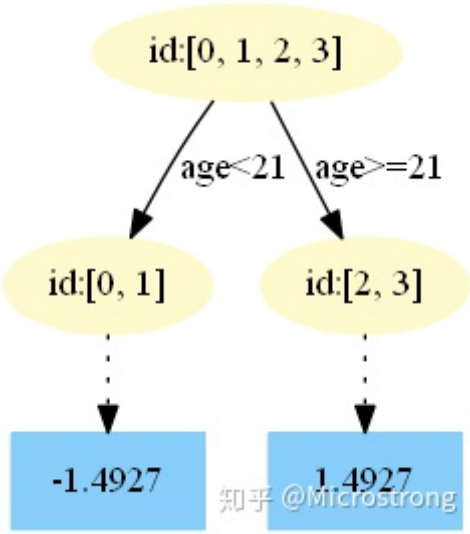
第三棵树：



第四棵树：



第五棵树：



3）得到最后的强学习器：

$$F_5(x) = F_0(x) + \sum_{m=1}^{5}\sum_{j=1}^{2} c_{m,j} I\left(x \in R_{m,j}\right)$$

（3）模型预测阶段

- $F_0(x) = 0$
- 在 $F_1(x)$ 中，测试样本的年龄为25，大于划分节点21岁，所以被预测为**2.0000**。
- 在 $F_2(x)$ 中，测试样本的年龄为25，大于划分节点21岁，所以被预测为**1.8187**。
- 在 $F_3(x)$ 中，测试样本的年龄为25，大于划分节点21岁，所以被预测为**1.6826**。
- 在 $F_4(x)$ 中，测试样本的年龄为25，大于划分节点21岁，所以被预测为**1.5769**。
- 在 $F_5(x)$ 中，测试样本的年龄为25，大于划分节点21岁，所以被预测为**1.4927**。

最终预测结果为：

$$F(x) = 0.0000 + 0.1 * (2.0000 + 1.8187 + 1.6826 + 1.5769 + 1.4927) = 0.8571$$

$$P(Y=1|x) = \frac{1}{1+e^{-F(x)}} = \frac{1}{1+e^{-0.8571}} = 0.7021$$