深入理解GBDT回归算法

原创 Microstrong Microstrong 2019-10-26

收录于话题
#Boosting算法总结

8个

目录：

本文的主要内容概览：

1. GBDT简介

Boosting、Bagging和Stacking是集成学习(Ensemble Learning)的三种主要方法。Boosting是一族可将弱学习器提升为强学习器的算法，不同于Bagging、Stacking方法，Boosting训练过程为串联方式，弱学习器的训练是有顺序的，每个弱学习器都会在前一个学习器的基础上进行学习，最终综合所有学习器的预测值产生最终的预测结果。

梯度提升（Gradient Boosting）算法是一种用于回归、分类和排序任务的机器学习技术，属于Boosting算法族的一部分。之前我们介绍过Gradient Boosting算法在迭代的每一步构建一个能够沿着梯度最陡的方向降低损失的学习器来弥补已有模型的不足。经典的AdaBoost算法只能处理采用指数损失函数的二分类学习任务，而梯度提升方法通过

设置不同的可微损失函数可以处理各类学习任务（多分类、回归、Ranking等），应用范围大大扩展。梯度提升算法利用损失函数的负梯度作为残差拟合的方式，如果其中的基函数采用决策树的话，就得到了梯度提升决策树 (Gradient Boosting Decision Tree, GBDT)。

基于梯度提升算法的学习器叫做GBM(Gradient Boosting Machine)。理论上，GBM可以选择各种不同的学习算法作为基学习器。现实中，用得最多的基学习器是决策树。

决策树有以下优点：

- 决策树可以认为是if-then规则的集合，易于理解，可解释性强，预测速度快。
- 决策树算法相比于其他的算法需要更少的特征工程，比如可以不用做特征标准化。
- 决策树可以很好的处理字段缺失的数据。
- 决策树能够自动组合多个特征，也有特征选择的作用。
- 对异常点鲁棒
- 可扩展性强，容易并行。

决策树有以下缺点：

- 缺乏平滑性（回归预测时输出值只能输出有限的若干种数值）。
- 不适合处理高维稀疏数据。
- 单独使用决策树算法时容易过拟合。

我们可以通过抑制决策树的复杂性，降低单棵决策树的拟合能力，再通过梯度提升的方法集成多个决策树，最终能够很好的解决过拟合的问题。由此可见，梯度提升方法和决策树学习算法可以互相取长补短，是一对完美的搭档。

2. GBDT回归算法

2.1 GBDT回归算法推导

当我们采用的基学习器是决策树时，那么梯度提升算法就具体到了梯度提升决策树。GBDT算法又叫MART（Multiple Additive Regression），是一种迭代的决策树算法。GBDT算法可以看成是 M 棵树组成的加法模型，其对应的公式如下：

$$F(x,w)=\sum_{m=0}^{M}\alpha_m h_m(x,w_m)=\sum_{m=0}^{M}f_m(x,w_m)$$

其中，x为输入样本；w为模型参数；h为分类回归树；α为每棵树的权重。GBDT算法的实现过程如下：

给定训练数据集：$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$ 其中，$x_i \in \chi \subseteq R^n$，$\chi$为输入空间，$y_i \in Y \subseteq R$，$Y$为输出空间，损失函数为$L(y,f(x))$，我们的目标是得到最终的回归树$F_M$。

（1）初始化第一个弱学习器$F_0(x)$：

$$F_0(x)=\arg\min_c \sum_{i=1}^{N} L\left(y_i,c\right)$$

（2）对于建立M棵分类回归树$m=1,2,\ldots,M$：

a) 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）：

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

b) 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。

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：

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

d) 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：

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

2.2 GBDT回归算法实例

（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，身高为标签值，共有4组数据。

编号	年龄(岁)	体重(kg)	身高(m)(标签)
0	5	20	1.1
1	7	30	1.3
2	21	70	1.7
3	30	60	1.8

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值为多少。

编号	年龄(岁)	体重(kg)	身高(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：

$$F_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

损失函数为平方损失，因为平方损失函数是一个凸函数，可以直接求导，令导数等于零，得到 c。

$$\sum_{i=1}^{N} \frac{\partial L(y_i, c)}{\partial c} = \sum_{i=1}^{N} \frac{\partial (\frac{1}{2}(y_i - c)^2)}{\partial c} = \sum_{i=1}^{N} c - y_i$$

令导数等于0：

$$\sum_{i=1}^{N} c - y_i = 0 \Rightarrow c = \frac{\sum_{i=1}^{N} y_i}{N}$$

所以初始化时，c 取值为所有训练样本标签值的均值。$c = (1.1 + 1.3 + 1.7 + 1.8)/4 = 1.475$，此时得到的初始化学习器为 $F_0(x) = c = 1.475$。

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平方损失时，负梯度就是残差，也就是 y 与上一轮得到的学习器 F_{m-1} 的差值：

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

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

此时将残差作为样本的真实值来训练弱学习器$F_1(x)$，即下表数据：

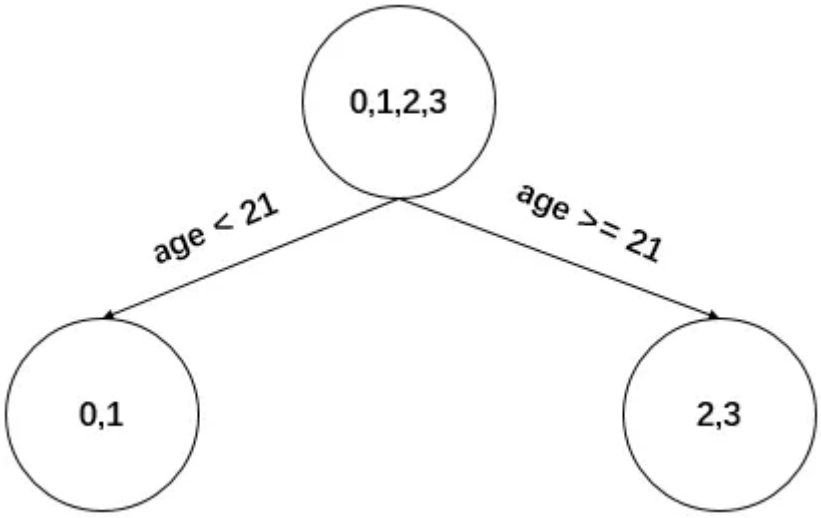
编号	年龄(岁)	体重(kg)	标签值
0	5	20	-0.375
1	7	30	-0.175
2	21	70	0.225
3	30	60	0.325

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，则$SE_l = 0$、$SE_r = 0.140$、$SE_{sum} = 0.140$，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	0.327	0.327
年龄7	0	1，2，3	0	0.140	0.140
年龄21	0，1	2，3	0.020	0.005	**0.025**
年龄30	0，1，2	3	0.187	0	0.187
体重20	/	0，1，2，3	0	0.327	0.327
体重30	0	1，2，3	0	0.140	0.140
体重60	0，1	2，3	0.020	0.005	**0.025**
体重70	0，1，3	2	0.260	0	0.260

以上划分点的总平方损失最小为**0.025**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：

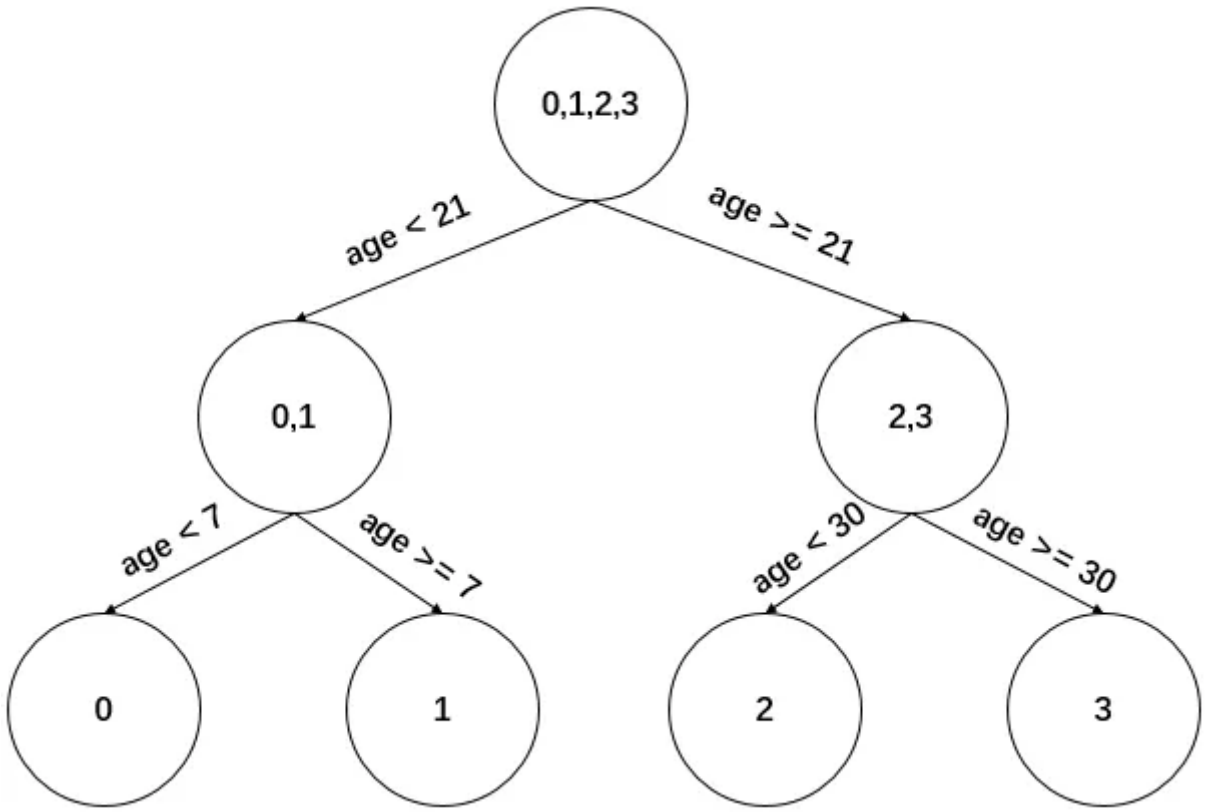
对于左节点，只含有0,1两个样本，根据下表结果我们选择**年龄7**为划分点（也可以选体重30）。

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	0	0.020	0.020
年龄7	0	1	0	0	**0**
体重20	/	0，1	0	0.020	0.020
体重30	0	1	0	0	**0**

对于右节点，只含有2,3两个样本，根据下表结果我们选择**年龄30**为划分点（也可以选体重70）。

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21	/	2，3	0	0.005	0.005
年龄30	2	3	0	0	**0**
体重60	/	2，3	0	0.005	0.005
体重70	3	2	0	0	**0**

现在我们的第一棵回归树长下面这个样子：



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

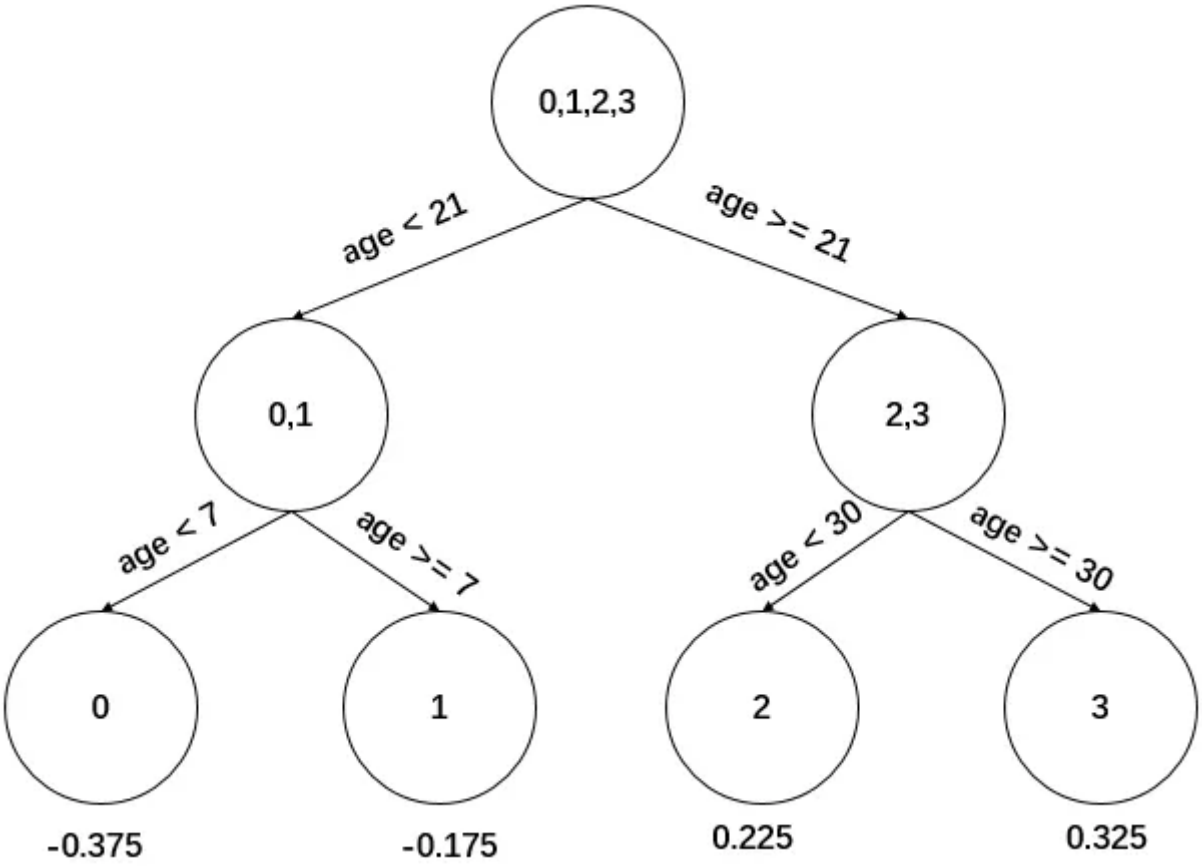
$$c_{1,j} = \arg\min_{c} \sum_{x_i \in R_{1,j}} L\left(y_i, F_0\left(x_i\right) + c\right)$$

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$y - f_0(x)$。

根据上述划分结果，为了方便表示，规定从左到右为第1,2,3,4个叶子结点，其计算值过程如下：

$$\begin{aligned}
&(x_0 \in R_{1,1}), \qquad c_{1,1} = 1.1 - 1.475 = -0.375 \\
&(x_1 \in R_{1,2}), \qquad c_{1,2} = 1.3 - 1.475 = -0.175 \\
&(x_2 \in R_{1,3}), \qquad c_{1,3} = 1.7 - 1.475 = 0.225 \\
&(x_3 \in R_{1,4}), \qquad c_{1,4} = 1.8 - 1.475 = 0.325
\end{aligned}$$

此时的树长这下面这个样子：



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。更新公式为：

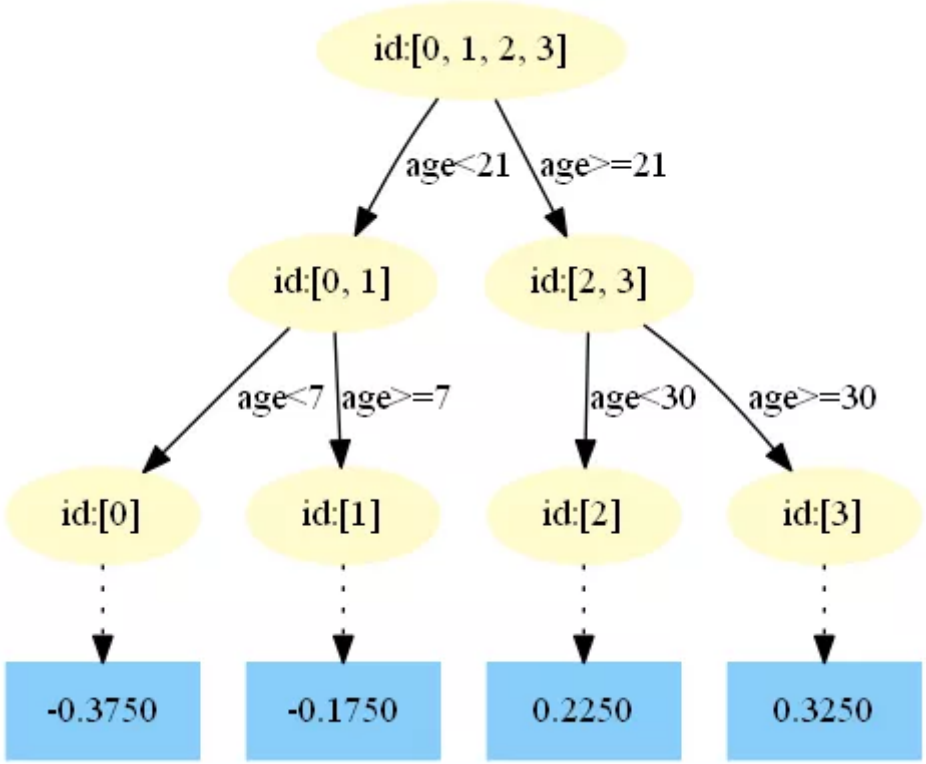
$$F_1(x) = F_0(x) + lr * \sum_{j=1}^{4} c_{1,j} I\left(x \in R_{1,j}\right)$$

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。

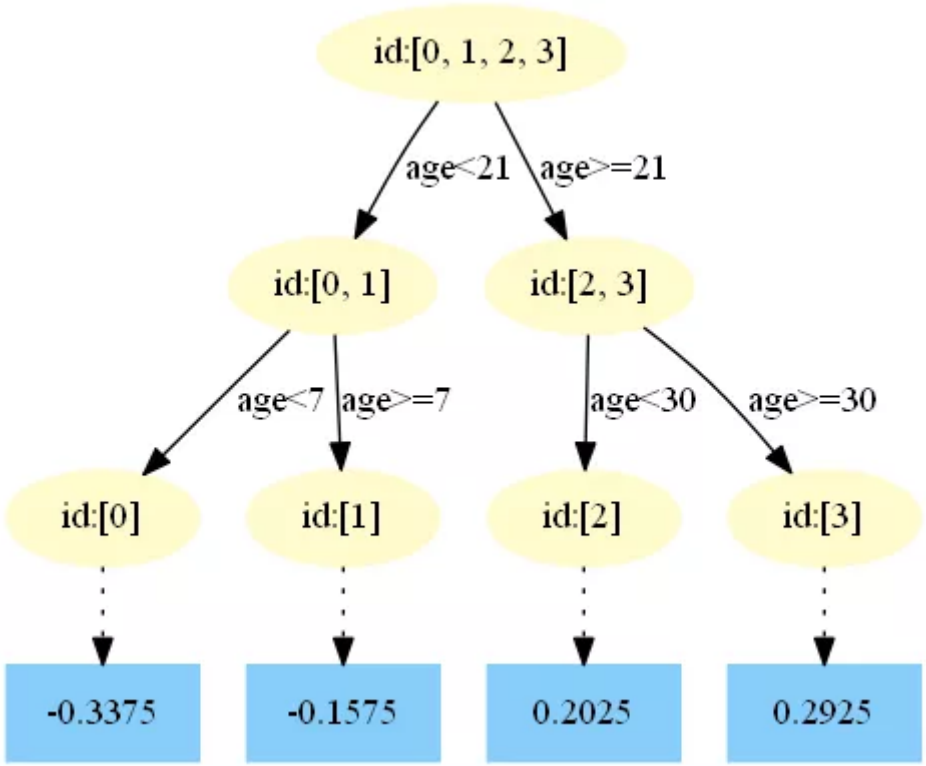
重复此步骤，直到 $m > 5$ 结束，最后生成5棵树。

下面将展示每棵树最终的结构，这些图都是我GitHub上用Python3实现GBDT代码生成的，感兴趣的同学可以去运行一下代码。地址：https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/tree/master/Ensemble%20Learning/GBDT_Regression

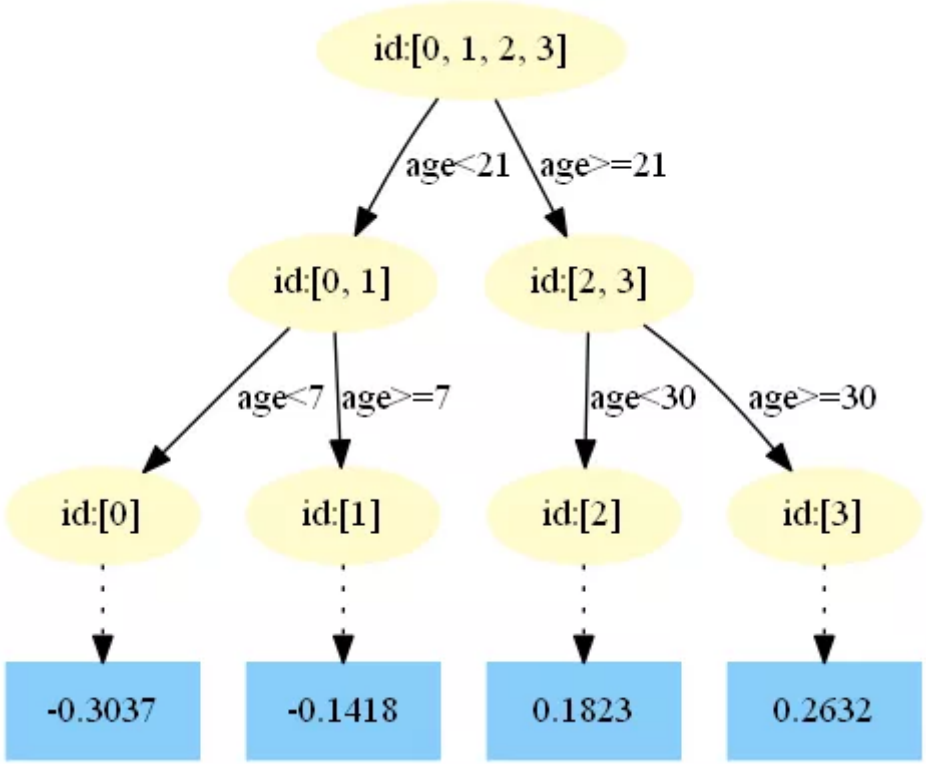
第一棵树：


id:[0, 1, 2, 3]
age<21
age>=21
id:[0, 1]
id:[2, 3]
age<7
age>=7
age<30
age>=30
id:[0]
id:[1]
id:[2]
id:[3]
-0.3750
-0.1750
0.2250
0.3250

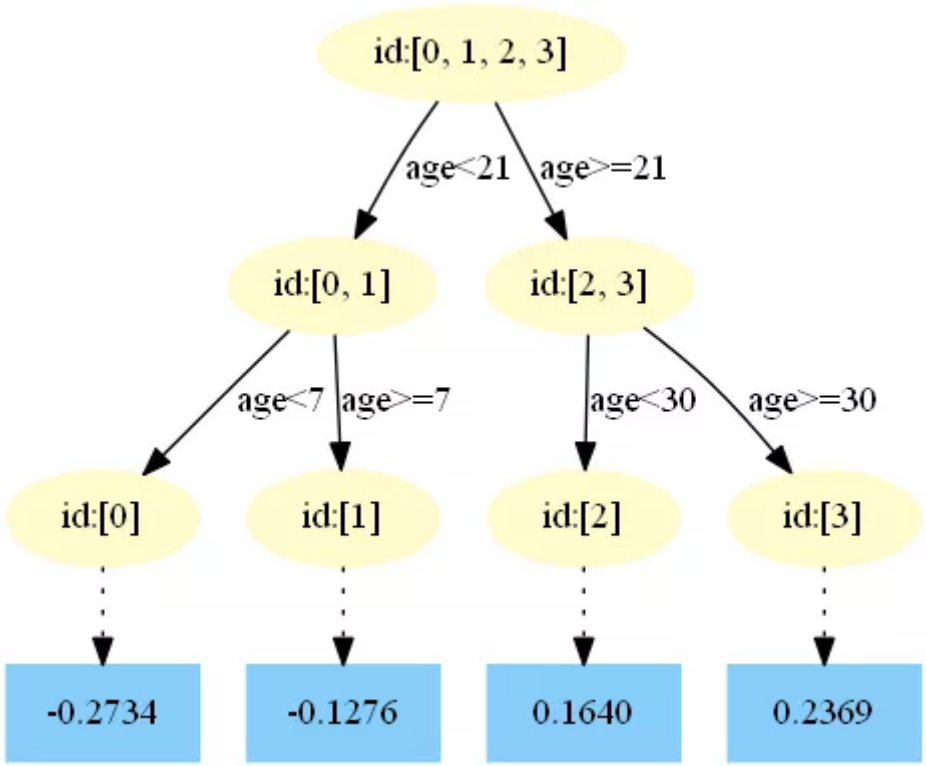
第二棵树：


id:[0, 1, 2, 3]
age<21
age>=21
id:[0, 1]
id:[2, 3]
age<7
age>=7
age<30
age>=30
id:[0]
id:[1]
id:[2]
id:[3]
-0.3375
-0.1575
0.2025
0.2925

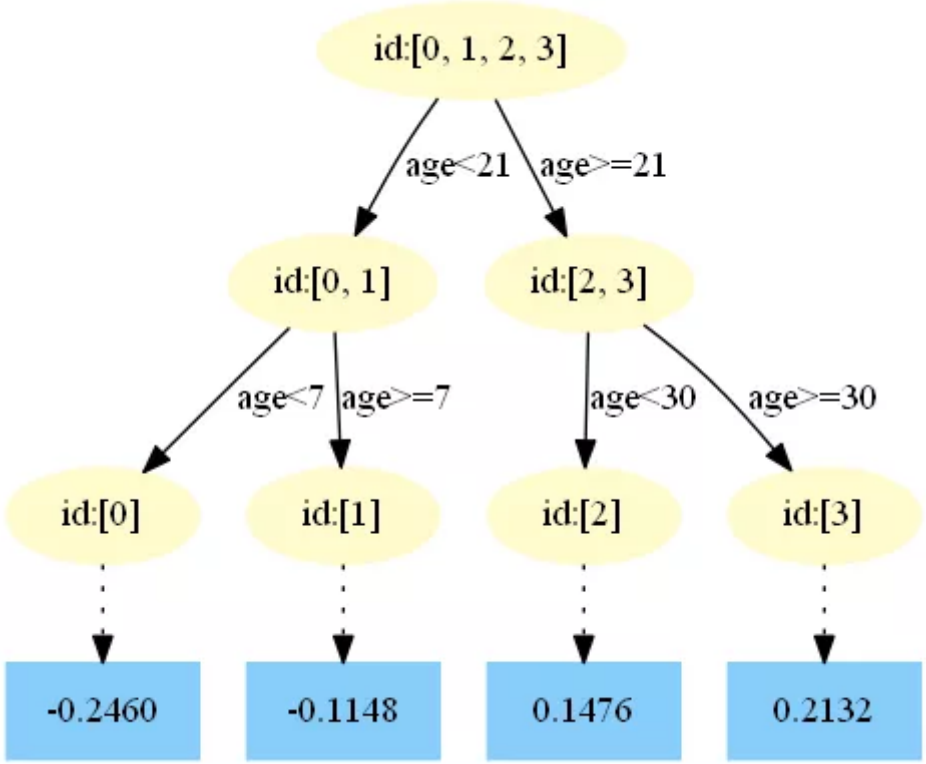
第三棵树：


id:[0, 1, 2, 3]
age<21
age>=21
id:[0, 1]
id:[2, 3]
age<7
age>=7
age<30
age>=30
id:[0]
id:[1]
id:[2]
id:[3]
-0.3037
-0.1418
0.1823
0.2632

第四棵树：


id:[0, 1, 2, 3]
age<21
age>=21
id:[0, 1]
id:[2, 3]
age<7
age>=7
age<30
age>=30
id:[0]
id:[1]
id:[2]
id:[3]
-0.2734
-0.1276
0.1640
0.2369

第五棵树：



3）得到最后的强学习器：

$$F_5(x) = F_0(x) + \sum_{m=1}^{5}\sum_{j=1}^{4} c_{m,j} I\left(x \in R_{m,j}\right)$$

（3）模型预测阶段

- $F_0(x) = 1.475$
- 在$F_1(x)$中，测试样本的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.2250**。
- 在$F_2(x)$ 中，测试样本的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.2025**。
- 在 $F_3(x)$中，测试样本的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.1823**。
- 在$F_4(x)$中，测试样本的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.1640**。
- 在$F_5(x)$中，测试样本的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.1476**。

最终预测结果为：

$$F(x) = 1.475 + 0.1 * (0.225 + 0.2025 + 0.1823 + 0.164 + 0.1476) = 1.56714$$

3. 手撕GBDT回归算法

本篇文章所有数据集和代码均在我的GitHub中，地址：https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/tree/master/Ensemble%20Learning

3.1 用Python3实现GBDT回归算法

需要的Python库：

```
pandas、PIL、pydotplus、matplotlib
```

其中pydotplus库会自动调用Graphviz，所以需要去Graphviz官网下载graphviz-2.38.msi安装，再将安装目录下的bin添加到系统环境变量，最后重启计算机。

由于用Python3实现GBDT回归算法代码量比较多，我这里就不列出详细代码了，感兴趣的同学可以去我的GitHub中看一下，地址：https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/tree/master/Ensemble%20Learning/GBDT_Regression

3.2 用sklearn实现GBDT回归算法

```
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor

gbdt = GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=5,
                                 , min_samples_split=2, min_samples_leaf=1, ma
                                 , init=None, random_state=None, max_features=
                                 , alpha=0.9, verbose=0, max_leaf_nodes=None
                                 , warm_start=False
                                 )
train_feat = np.array([[1, 5, 20],
                       [2, 7, 30],
                       [3, 21, 70],
```

```
                        [4, 30, 60],
                        ])
train_id = np.array([[1.1], [1.3], [1.7], [1.8]]).ravel()
test_feat = np.array([[5, 25, 65]])
test_id = np.array([[1.6]])
print(train_feat.shape, train_id.shape, test_feat.shape, test_id.shape)
gbdt.fit(train_feat, train_id)
pred = gbdt.predict(test_feat)
total_err = 0
for i in range(pred.shape[0]):
    print(pred[i], test_id[i])
    err = (pred[i] - test_id[i]) / test_id[i]
    total_err += err * err
print(total_err / pred.shape[0])
```

用sklearn中的GBDT库实现GBDT回归算法的难点在于如何更好的调节下列参数：

```
_SUPPORTED_LOSS = ('ls', 'lad', 'huber', 'quantile')
```

```
def __init__(self, loss='ls', learning_rate=0.1, n_estimators=100,
             subsample=1.0, criterion='friedman_mse', min_samples_split=2,
             min_samples_leaf=1, min_weight_fraction_leaf=0.,
             max_depth=3, min_impurity_decrease=0.,
             min_impurity_split=None, init=None, random_state=None,
             max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None,
             warm_start=False, presort='auto', validation_fraction=0.1,
             n_iter_no_change=None, tol=1e-4):
```

用sklearn实现GBDT回归算法的GitHub地址：https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/tree/master/Ensemble%20Learning/GBDT_Regression_sklearn

4. GBDT回归任务常见的损失函数

对于GBDT回归模型，sklearn中实现了四种损失函数，有均方差'ls'，绝对损失'lad'，Huber损失'huber'和分位数损失'quantile'。默认是均方差'ls'。一般来说，如果数据的噪音点不多，用默认的均方差'ls'比较好。如果是噪音点较多，则推荐用抗噪音的损失函

数'huber'。而如果我们需要对训练集进行分段预测的时候，则采用'quantile'。下面我们具体来了解一下这四种损失函数。

（1）均方差，这个是最常见的回归损失函数了，公式如下：

$$L(y, f(x)) = (y - f(x))^2$$

对应的负梯度误差为：

$$y_i - f(x_i)$$

（2）绝对损失，这个损失函数也很常见，公式如下：

$$L(y, f(x)) = |y - f(x)|$$

对应的负梯度误差为：

$$sign(y_i - f(x_i))$$

（3）Huber损失，它是均方差和绝对损失的折衷产物，对于远离中心的异常点，采用绝对损失，而中心附近的点采用均方差。这个界限一般用分位数点度量。损失函数如下：

$$L(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & |y - f(x) \le \delta| \\ \delta(|y - f(x)| - \frac{\delta}{2}) & |y - f(x) > \delta| \end{cases}$$

对应的负梯度误差为：

$$r(y_i, f(x_i)) = \begin{cases} y_i - f(x_i) & |y_i - f(x_i) \le \delta| \\ \delta \cdot sign(y_i - f(x_i)) & |y_i - f(x_i) > \delta| \end{cases}$$

（4）分位数损失，它对应的是分位数回归的损失函数，表达式为：

$$L(y, f(x)) = \sum_{y \ge f(x)} \theta \, |y - f(x)| + \sum_{y < f(x)} (1 - \theta) \, |y - f(x)|$$

其中，θ为分位数，需要我们在回归前指定。对应的负梯度误差为：

$$r(y_i, f(x_i)) = \begin{cases} \theta & y_i \geq f(x_i) \\ \theta - 1 & y_i < f(x_i) \end{cases}$$

对于Huber损失和分位数损失，主要用于健壮回归，也就是减少异常点对损失函数的影响。

5. GBDT的正则化

为了防止过拟合，GBDT主要有五种正则化的方式。

（1）“Shrinkage”：这是一种正则化（regularization）方法，为了防止过拟合，在每次对残差估计进行迭代时，不直接加上当前步所拟合的残差，而是乘以一个系数α。系数α也被称为学习率（learning rate），因为它可以对梯度提升的步长进行调整，也就是它可以影响我们设置的回归树个数。对于前面的弱学习器的迭代：

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

如果我们加上了正则化项，则有：

$$F_m(x) = F_{m-1}(x) + \alpha h_m(x)$$

α的取值范围为 $0 < \alpha \leq 1$。对于同样的训练集学习效果，较小的 α意味着我们需要更多的弱学习器的迭代次数。通常我们用学习率和迭代最大次数一起来决定算法的拟合效果。即参数learning_rate会强烈影响到参数n_estimators（即弱学习器个数）。learning_rate的值越小，就需要越多的弱学习器数来维持一个恒定的训练误差(training error)常量。经验上，推荐小一点的learning_rate会对测试误差(test error)更好。在实际调参中推荐将learning_rate设置为一个小的常数（e.g. learning_rate <= 0.1），并通过early stopping机制来选n_estimators。

（2）“Subsample”：第二种正则化的方式是通过子采样比例（subsample），取值为 (0,1]。注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但会增加样本拟合的偏差，因此取值不能太低。推荐在 [0.5, 0.8]之间。

使用了子采样的GBDT有时也称作随机梯度提升树 (Stochastic Gradient Boosting Tree, SGBT)。由于使用了子采样，程序可以通过采样分发到不同的任务去做Boosting的迭代过程，最后形成新树，从而减少弱学习器难以并行学习的弱点。

（3）对于弱学习器即CART回归树进行正则化剪枝。这一部分在学习决策树原理时应该掌握的，这里就不重复了。

（4）“Early Stopping”：Early Stopping是机器学习迭代式训练模型中很常见的防止过拟合技巧，具体的做法是选择一部分样本作为验证集，在迭代拟合训练集的过程中，如果模型在验证集里错误率不再下降，就停止训练，也就是说控制迭代的轮数（树的个数）。在sklearn的GBDT中可以设置参数n_iter_no_change实现early stopping。

（5）“Dropout”：Dropout是deep learning里很常用的正则化技巧，很自然的我们会想能不能把Dropout用到GBDT模型上呢？AISTATS2015有篇文章《DART: Dropouts meet Multiple Additive Regression Trees》进行了一些尝试。文中提到GBDT里会出现over-specialization的问题：前面迭代的树对预测值的贡献比较大，后面的树会集中预测一小部分样本的偏差。Shrinkage可以减轻over-specialization的问题，但不是很好。作者想通过Dropout来平衡所有树对预测的贡献。

具体的做法是：每次新加一棵树，这棵树要拟合的并不是之前全部树ensemble后的残差，而是随机抽取的一些树ensemble；同时新加的树结果要规范化一下。对这一部分感兴趣的同学可以阅读一下原论文。

6. 关于GBDT若干问题的思考

（1）GBDT与AdaBoost的区别与联系？

AdaBoost和GBDT都是重复选择一个表现一般的模型并且每次基于先前模型的表现进行调整。不同的是，AdaBoost是通过调整错分数据点的权重来改进模型，GBDT是通过计算负梯度来改进模型。因此，相比AdaBoost, GBDT可以使用更多种类的目标函数，而当目标函数是均方误差时，计算损失函数的负梯度值在当前模型的值即为残差。

（2）GBDT与随机森林（Random Forest，RF）的区别与联系？

相同点：都是由多棵树组成，最终的结果都是由多棵树一起决定。

不同点：1）集成的方式：随机森林属于Bagging思想，而GBDT是Boosting思想。2）偏差-方差权衡：RF不断的降低模型的方差，而GBDT不断的降低模型的偏差。3）训

练样本方式：RF每次迭代的样本是从全部训练集中有放回抽样形成的，而GBDT每次使用全部样本。4）并行性：RF的树可以并行生成，而GBDT只能顺序生成(需要等上一棵树完全生成)。5）最终结果：RF最终是多棵树进行多数表决（回归问题是取平均），而GBDT是加权融合。6）数据敏感性：RF对异常值不敏感，而GBDT对异常值比较敏感。7）泛化能力：RF不易过拟合，而GBDT容易过拟合。

（3）我们知道残差=真实值-预测值，明明可以很方便的计算出来，为什么GBDT的残差要用负梯度来代替？为什么要引入麻烦的梯度？有什么用呢？

回答第一小问：在GBDT中，无论损失函数是什么形式，每个决策树拟合的都是负梯度。准确的说，不是用负梯度代替残差，而是当损失函数是均方损失时，负梯度刚好是残差，残差只是特例。

回答二、三小问：GBDT的求解过程就是梯度下降在函数空间中的优化过程。在函数空间中优化，每次得到增量函数，这个函数就是GBDT中一个个决策树，负梯度会拟合这个函数。要得到最终的GBDT模型，只需要把初始值或者初始的函数加上每次的增量即可。我这里高度概括的回答了这个问题，详细推理过程可以参考：梯度提升（Gradient Boosting）算法，地址：https://mp.weixin.qq.com/s/Ods1PHhYyjkRA8bS16OfCg

7. 总结

在本文中，我们首先引出回归树与梯度提升算法结合的优势；然后详细推导了GBDT回归算法的原理，并用实际案例解释GBDT回归算法；其次不仅用Python3实现GBDT回归算法，还用sklearn实现GBDT回归算法；最后，介绍了GBDT回归任务常见的损失函数、GBDT的正则化和我对GBDT回归算法若干问题的思考。GBDT中的树是回归树（不是分类树），GBDT可以用来做回归预测，这也是我们本文讲的GBDT回归算法，但是GBDT调整后也可以用于分类任务。让我们期待一下GBDT分类算法，在分类任务中的表现吧！

8. Reference

由于参考的文献较多，我把每一部分都重点参考了哪些文章详细标注一下。

GBDT简介与GBDT回归算法：

【1】Friedman J H . Greedy Function Approximation: A Gradient Boosting Machine[J]. The Annals of Statistics, 2001, 29(5):1189-1232.