传统推荐算法(五) LR+GBDT (1) 剑指GBDT

原创 如雨星空 推荐算法工程师 2019-08-22

文章目录

写在前面

> 学习GBDT的时候，被网上的几篇文章搞晕了，就去看了下GBDT的论文，整理了一些思路，结合参考中的一些内容，整理了这篇文章。本文将循序渐进，从GB,DT讲到GBDT，细致分析下GBDT的原理。本人才疏学浅，有些地方可能理解得不对，欢迎指出错误。学习过程中，薛大佬的这篇文章给了我很多启发：http://xtf615.com/paper/GBM.html。他本人也很热心地帮我解答疑问，在此特别感谢。
>
> 机器学习中的 Boosting 算法族有两大类，一类是 weightboosting，其中以 adaboost 为主要代表，另一类是 gradientboosting，其中以 gbdt 为主要代表[1]。GBDT是机器学习竞赛中常用的一种算法，据统计,Kaggle比赛中50%以上的冠军方案都是基于GBDT算法[2]。有人称之为机器学习TOP3算法。1999年，Jerome Harold Friedman将GBDT算法用于回归分析并正式提出，通过改进也可以用于分类问题。实际上，GBDT可以进行分类，回归预测，还可以做排序任务（比如搜索引擎

> 排序）[4]。GBDT的论文地址为：
>
> https://projecteuclid.org/download/pdf_1/euclid.aos/1013203451
>
> GBDT到底是什么样子呢？阅读本文后，读者应该可以尝试回答出以下问题：
>
> - GBDT为什么使用负梯度？
> - GBDT为什么选择决策树(DT)为基学习器？
> - GBDT的算法流程？
> - GBDT如何选择特征？
> - 建树过程中，计算划分误差的叶节点值是否就是回归树的输出值？
> - 为什么有人说GBDT是在拟合残差？
> - GBDT如何进行分类？
> - GBDT的优缺点？

1. GBM

梯度提升（Gradient boosting）是一种用于回归、分类和排序任务的机器学习技术，是boosting族算法的一种。通过分步迭代（stage-wise）的方式来构建模型，在迭代的每一步构建的弱学习器都是为了弥补已有模型的不足[3]。最后将所有的弱学习器结合起来，得到一个强学习器。基于梯度提升算法的学习器叫做GBM(Gradient Boosting Machine)。

1.1 从参数空间到函数空间

这部分讲GB的思想动机，主要是论文第一、二节的内容。在看GBM之前，首先看参数空间一个典型的优化问题[7]：

$$\hat{x} = arg\min_{x} f(x)$$

对于这种问题，有一种解法是梯度下降法(Steepest Gradient Descent,SGD)，采用分布加和扩展的方案：

- 给定一个起始点x_0
- 对$i = 1, 2, \ldots, n$分别作如下迭代：
- $x_i = x_{i-1} + \beta_i * g_i$, 其中$g_i = -\frac{\partial f}{\partial x}|_{x=x_{i-1}}$表示$f$在$x_{i-1}$上的负梯度值，$\beta_i$是步长，是通过在$g_i$方向线性搜索来动态调整的。
- 一直到$|g_i|$足够小，或者$|x_i - x_{i-1}|$足够小，即函数收敛。

其实就是给定起点后，贪心寻找最优解。这里的贪心是指每步都是在上一步的基础上往函数下降最快的方向走。寻得的解可以表示为：

$$x_k = x_0 + \beta_1 * g_1 + \beta_2 * g_2 + \cdots + \beta_k * g_k$$

这是在参数空间进行最优参数点估计，这个思路能不能推广到函数空间，进行最优函数估计呢？首先，看个函数空间一般函数估计问题。函数估计的目标是得到Φ(F)，使得所有训练样本在(y,x)的联合分布上，最小化期望损失函数[3]：

$$\Phi(F) = E_{y,\mathbf{x}} L(y, F(\mathbf{x})) = E_{\mathbf{x}}[E_y(L(y, F(\mathbf{x}))) \mid \mathbf{x}],$$

在函数空间，理想状态下有无数个点，可以使用非参数方法解决以上问题，也就是不固定F(x)的形式。非参数方法不固定参数形式，实际上是有无限个参数。参考之前的前向加和扩展方案，解的形式可以表达为：

$$F^*(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x}),$$

其中$f_0(x)$是初始估计，$f_m(x)$是提升函数。而梯度下降的形式就是：

$$f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x})$$

其中：

$$g_m(\mathbf{x}) = \left[\frac{\partial \phi(F(\mathbf{x}))}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left[\frac{\partial E_y[L(y, F(\mathbf{x})) \mid \mathbf{x}]}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

$$F_{m-1}(\mathbf{x}) = \sum_{i=0}^{m-1} f_i(\mathbf{x}).$$

假设可以交换微分和积分的顺序，则有：

$$g_m(\mathbf{x}) = E_y\left[\frac{\partial L(y, F(\mathbf{x}))}{\partial F(\mathbf{x})} \mid \mathbf{x}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}.$$

$$\rho_m = \arg\min_{\rho} E_{y,\mathbf{x}} L(y, F_{m-1}(\mathbf{x}) - \rho g_m(\mathbf{x})).$$

作者将前向加和扩展与梯度下降的结合，成功应用到参数空间中，并给出非参估计形式下的解。

1.2 从非参估计到参数估计

然而论文指出，使用非参数估计时，模型很难被很好地准确评估。关于这一点，薛大佬认为：函数空间可以拟合原始数据的函数非常多，“可以有很多函数簇能够拟合原始训练数据，某个样本需要哪个函数簇来拟合是概率依赖的。”到底哪种组合才是最优的呢？这是个NP难问题了。非参估计F(x)不确定，参数就是无限的，很难直接求出最优拟合函数。因此作者给F(x)加上一些限制，假设一种参数化的形式，然后使用参数优化方法求解模型。

> 需要注意的是，这是把函数空间的优化从无参学习方法转为参数学习方法进行，而整体GB仍是在函数空间进行优化（观察每次> 损失函数优化的方向，是更新函数），只不过每一步的函数优化采用参数学习方法，在参数空间进行。

加上限制条件，目标函数可以写成如下的加和模型：

$$F(\mathbf{x};\{\beta_m,\mathbf{a}_m\}_1^M)=\sum_{m=1}^{M}\beta_m h(\mathbf{x};\mathbf{a}_m).$$

其中h(x)为基函数，a_m是基函数的参数，β_m可以看成是该学习器的权重。求解目标可以转化为以下形式：

$$\{\beta_m,\mathbf{a}_m\}_1^M=\arg\min_{\{\beta'_m,\mathbf{a}'_m\}_1^M}\sum_{i=1}^{N}L\left(y_i,\sum_{m=1}^{M}\beta'_m h(\mathbf{x}_i;\mathbf{a}'_m)\right).$$

boosting中，上述求解采取贪心分布迭代(Greedy Stagewise)，对m=1,2,...,M：

$$(\beta_m,\mathbf{a}_m)=\arg\min_{\beta,\mathbf{a}}\sum_{i=1}^{N}L(y_i,F_{m-1}(\mathbf{x}_i)+\beta h(\mathbf{x}_i;\mathbf{a})) \tag{9}$$

然后再更新：

$$F_m(\mathbf{x})=F_{m-1}(\mathbf{x})+\beta_m h(\mathbf{x};\mathbf{a}_m). \tag{10}$$

我们通过M步迭代，不断最小化损失函数，使得强分类器性能越来越优，强分类器$F_m(x)$最终进化为F(x)。

如果对于某个损失函数L(y,F)和基学习器h(x;a)，通过公式(9)很难直接求出其解析解，那该如何是好？

我们换个角度来看，对于给定的损失函数L(y,F)，迭代求解基学习器时，损失函数不断变小越小，F_m也就越靠近F，结合公式10，**$\beta_m h(x;a_m)$可以看成是已知$F_{m-1}(x)$情况下，向最终解F(x)一步最优贪心靠拢，其中$h(x;a_m)$无疑给我们指明了靠拢的方向**。

而当前步要求解的$\beta_m h(x;a_m)$，要尽可能使得损失函数最小化——**也就是使得L(y,F)在上一步的基础上快速下降，其中$h(x;a_m)$就间接决定了下降的方向**。使得损失函数极速下降的最优方向，就是负梯度：

$$-g_m(\mathbf{x}_i)=-\left[\frac{\partial L(y_i,F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

这是βmh(x;am)能达到的最优值，也就是βmh(x;am)要拟合的值。参数求解问题就转化为了每一步βmh(x;am)与最优值的拟合问题，很难直接求出解析解的问题就解决了。下面简单看下求解过程。首先对于这种函数拟合问题，可以使用平方误差进行拟合：

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [-g_m(\mathbf{x}_i) - \beta h(\mathbf{x}_i;\mathbf{a})]^2. \tag{11}$$

论文中提到，这个平方误差损失函数是一个常见的经验选择，但并不是唯一的选择。然后使用线性搜索方法搜索系数β:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i;\mathbf{a}_m)) \tag{12}$$

整个GB算法就是：

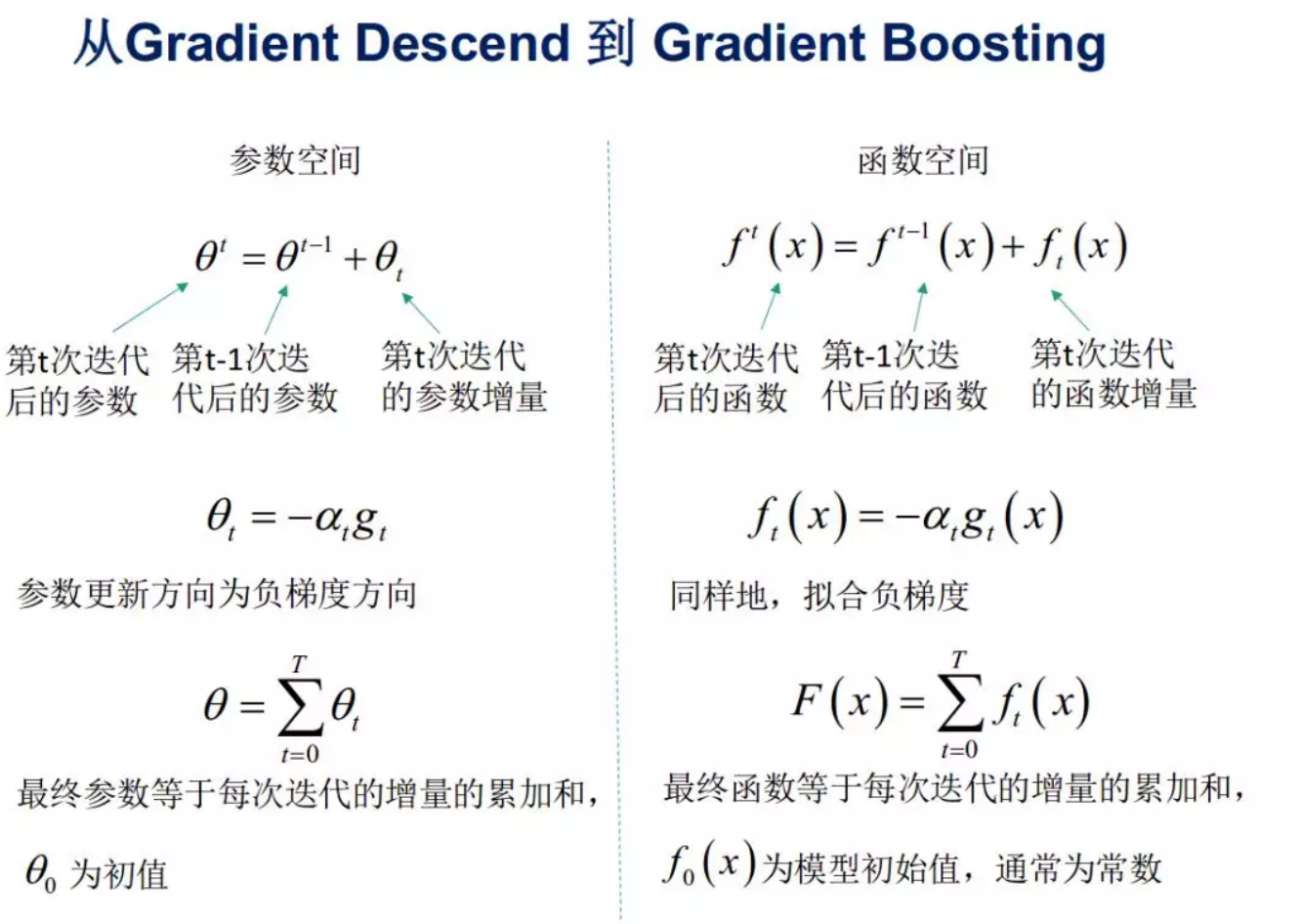
ALGORITHM 1 (Gradient_Boost).

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For $m = 1$ to M do:
3. $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i;\mathbf{a})]^2$
5. $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i;\mathbf{a}_m))$
6. $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x};\mathbf{a}_m)$

这里要注意，第4步是一个学习基学习器的过程。这里建树时节点由划分函数确定的拟合值，就直接作为回归树最终的拟合值。实际上我们使用GBDT时，使用平方误差函数拟合的回归树，其拟合值只是用来建树（有没有很迷），最终基学习器的输出值是根据损失函数来确定，和这里有些不一致。这是作者在GB的应用部分提出的一个改进策略，使用损失函数而非划分函数来确定输出值。**后文会详细讲一下这一点。**

其实仔细想想，这个和常规的的利用梯度下降最小化损失函数(比如cnn)过程一致。数值型解析解不能一步到位求出来，用梯度下降一步一步贪心近似；而我们的模型不能一步到位求出

来，就用boosting的方式一步一步地近似出理想模型。参数梯度下降是根据负梯度调整原来的参数，在参数层面调整；而此处根据负梯度，获得一个新的函数/基学习器（通过获得新的参数实现，但是参数的意义通过函数体现），在函数层面调整。虽然都是。来张图[5]感受一下：



1.3 泰勒展开近似

负梯度的理论支撑是泰勒公式的一阶展开近似[8]：

$$f(x+\Delta x) \approx f(x) + f'(x)\Delta x$$

对$L(y_i, F_{m-1}(x_i) + f_m(x_i))$作泰勒展开，有：

$L(y_i, F_{m-1}(x_i) + f_m(x_i)) = L(y_i, F_{m-1}(x_i)) + g_i * f_m(x_i)$

其中**g_i是$L(y_i, F_{m-1}(x_i))$**关于**$F_{m-1}(x_i)$**的一阶导，我们是希望损失函数是不断变小的，也就是：

$L(y_i, F_{m-1}(x_i) + f_m(x_i)) < L(y_i, F_{m-1}(x_i))$

所以关键是$g_i f_m(x_i)$这一项。然而我们不知道g_i是正还是负，那么只需要让$f_m(x_i) = -\alpha g_i$(α是任意正系数)，就可以让$g_i f_m(x_i)$恒为负了。

2. GBM的基学习器

2.1 基学习器选择

> 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。为什么梯度提升方法倾向于选择决策树（通常是CART树）作为基学习器呢？这与决策树算法自身的优点有很大的关系。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。同时，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，可以很好的处理字段缺失的数据，也可以不用关心特征间是否相互依赖等。决策树能够自动组合多个特征，它可以毫无压力地处理特征间的交互关系并且是非参数化的，因此你不必担心异常值或者数据是否线性可分（举个例子，决策树能轻松处理好类别A在某个特征维度x的末端，类别B在中间，然后类别A又出现在特征维度x前端的情况）。不过，单独使用决策树算法时，有容易过拟合缺点。所幸的是，通过各种方法，抑制决策树的复杂性，降低单颗决策树的拟合能力，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。至于抑制单颗决策树的复杂度的方法有很多，比如限制树的最大深度、限制叶子节点的最少样本数量、限制节点分裂时的最少样本数量、吸收bagging的思想对训练样本采样（subsample），在学习单颗决策树时只使用一部分训练样本、借鉴随机森林的思路在学习单颗决策树时只采样一部分特征、在目标函数中添加正则项惩罚复杂的树结构等[3]。

2.2 CART回归树

GBDT中的决策树，拟合的是负梯度，负梯度是连续值，所以用的是回归树，而非分类树。在这一点上，所以GBDT又常被叫做GBRT。而最常用的基学习器就是CART算法中的回归树。

分类与回归树(Classification and regression tree, CART)是Breiman等人在1984年提出的，可用于分类，也可用于回归。CART算法中的回归树生成过程通常称为最小二乘回归树生成算法：

输入：训练数据集D:

输出：回归树$f(x)$.

在训练数据集所在的输入空间中，递归的将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

（1）选择最优切分变量j与切分点s，求解

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

遍历变量j，对固定的切分变量j扫描切分点s，选择使得上式达到最小值的对(j,s).

（2）用选定的对(j,s)划分区域并决定相应的输出值：

$$R_1(j,s)=x|x^{(j)}\leq s, R_2(j,s)=x|x^{(j)}>s$$

$$\hat{c_m}=\frac{1}{N}\sum_{x_1\in R_m(j,s)}y_i, x\in R_m, m=1,2$$

（3）继续对两个子区域调用步骤（1）和（2），直至满足停止条件。

（4）将输入空间划分为M个区域$R_1,R_2,...R_M$，生成决策树：

$$f(x)=\sum_{m=1}^{M}\hat{c}_m I(x\in R_m)$$

其中的I函数为符号函数，满足括号中条件为1，否则为0。这个建树过程应该是很清晰的，切分函数为平方误差。

关于切分函数如何决定回归树的结构，我多扯两句。上述算法中划分函数是平方误差，**（1）中的c1和c2的计算公式是怎么来的呢**，取平方误差的最小值，c的取值就是区域上y_i的均值。也就是说根据划分函数来建树时，每个区域的输出值是根据划分函数计算的。

根据划分函数得到的输出值主要作用是用来建树，树的结构确定后，再极小化损失函数，得到叶子节点的输出值，这个输出值才是回归树的输出值。如果损失函数是平方误差，树节点的输出值恰好也是要拟合的y_i的均值。后面会结合公式详细讲解。

3．GBDT之回归

回归树有两个点需要注意，第一个是切分准则(fitting criterion/splitting criterion)，决定树的结构；另一个是损失函数(loss criterion)，决定树的输出。

之前提到，GB实际上只是一种策略，将分步加和扩展和梯度下降结合起来，实现对函数空间的数值优化。而当基分类器使用决策树DT时，就是我们常说的GBDT算法。假设我们选择的损失函数为：

$$L(y, F) = (y - F)^2/2.$$

论文中称之为least squares，比我们平时使用的和方差SSE多了一个1/2，显然求梯度时更方便。那么GBDT算法就可以表示为[10]：

Algorithm 2 : LS_TreeBoost

$$F_0(x) = \bar{y}$$

$$Form = 1 \ to \ M \ do :$$

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

$$\{R_{jm}\}_1^J = J - terminal \ node \ tree(\{\tilde{y}_i, x_i\}_1^N)$$

$$\gamma_{jm} = ave_{x_i \in R_{jm}} \tilde{y}_i$$

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$$

逐行分析。第一行是基学习器初始化。初始化怎么做的呢？其实是这样来的：

$$F_0(x) = argmix_\rho \sum_{i=1}^{N} L(y_i, \rho)$$

初始化估计使损失函数极小化的常数值，它是只有一个根节点的树。根节点的输出为N个y_i的均值。

第二行代表GBDT分布迭代总共迭代M次，也就是分步加和要加M步，每步都要训练一个基学习器。**第三行就是求负梯度了**，在上一步的基础上，得到损失函数在函数空间优化的局部最优方向。负梯度的取值和损失函数有关。当损失函数是least squares，负梯度为：

$$-\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right] = (y_i - F(x_i))$$

损失函数是绝对损失：

$$L(y, F) = |y - F|$$

负梯度就是：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \operatorname{sign}(y_i - F_{m-1}(\mathbf{x}_i)).$$

第四行意思就是把根据上轮结果算出的负梯度和样本x作为本轮树的输入，拟合一个回归树，并得到树的各节点划分属性、划分值等。这个拟合过程依据划分函数(fitting criterion/splitting criterion)确定，划分损失/划分质量是怎么计算的呢？输入(x,y)，其中y是计算出的负梯度，是被拟合值。而在回归树中，划分损失的计算还需要拟合值，拟合值是如何计算的呢？**这里简单提下，拟合值是根据划分函数计算，使得划分函数极小化的叶节点的值，就是拟合值。**而使用平方误差时，叶节点的拟合值就是叶节点区域所有样本负梯度的均值。

从另一个角度看，一般情况下，我们希望分支节点的样本尽可能属于同一类别，其方差尽可能小，所以拟合值就是叶节点区域所有样本负梯度的均值，拟合函数取方差。

第五行就是求叶子节点的取值。这个是很多人都忽略的一点！下面重点分析。叶子节点的取值和损失函数有关。使用"平方误差"时(SSE(CART树),MSE和least squares)，使树叶子节点的损失最小的取值就是:

$$\gamma_{jm} = ave_{x_i \in R_{jm}} \tilde{y}_i$$

根据论文中的内容，当损失函数是绝对误差(SAE,MAE)时，叶子节点取值就是：

$$\gamma_{jm} = \operatorname{median}_{\mathbf{x}_i \in R_{jm}} \{y_i - F_{m-1}(\mathbf{x}_i)\},$$

当损失函数是Huber loss时，叶子节点取值就是：

$$\gamma_{jm} = \tilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{\mathbf{x}_i \in R_{jm}} sign\left(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm}\right) \cdot \min(\delta_m, abs(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{jm}))$$

第六行就是前向加和，更新强学习器。在Scikit-learn的代码实现中，会人为地在弱学习器前面加个学习率，防止过拟合：

```
def update_terminal_regions(self, tree, X, y, residual, y_pred,
                            sample_weight, sample_mask,
```

```
                                learning_rate=1.0, k=0):

        # compute leaf for each sample in ``X``.
        terminal_regions = tree.apply(X)

        # mask all which are not in sample mask.
        masked_terminal_regions = terminal_regions.copy()
        masked_terminal_regions[~sample_mask] = -1

        # update each leaf (= perform line search)
        for leaf in np.where(tree.children_left == TREE_LEAF)[0]:
            self._update_terminal_region(tree, masked_terminal_regions,
                                         leaf, X, y, residual,
                                         y_pred[:, k], sample_weight)

        # update predictions (both in-bag and out-of-bag)
        y_pred[:, k] += (learning_rate
                         * tree.value[:, 0, 0].take(terminal_regions, axis=0))
```

GBDT最常用的决策树就是CART树，之所以不讲CART树版的GBDT，是因为CART树的划分函数和损失函数都可以看成SSE，建树过程的拟合值和叶节点的输出值是一致的，容易搞混。

4. GBDT之分类

GBDT由于拟合的是负梯度，所以用的是回归树，建树过程仍然是回归。建树完成后，最小化损失函数得到的预测值仍然是连续值。如何使用回归手段解决分类问题？之前的文章中我们提到过一个逻辑回归。实际上，GBDT做法类似，使用概率函数，将连续值转为概率，用对数似然损失来衡量预测值和真值间的误差。对这点不清楚可以参考之前的文章《传统推荐算法(五) FFM模型(1) 逻辑回归损失函数》。另外，[11]中Pinard大神指出，使用指数损失函数，此时GBDT退化为Adaboost算法，感兴趣可以自己深入研究。

4.1 二类逻辑回归和分类

论文中给了个二分类的例子，没有使用逻辑回归，使用的是负二项回归，其实差不多：

$$L(y, F) = \log(1 + \exp(-2yF)), \qquad y \in \{-1, 1\},$$

负梯度为：

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = 2y_i/(1 + \exp(2y_i F_{m-1}(\mathbf{x}_i))).$$

回归树叶子节点的预测值为：

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jm}} \log(1 + \exp(-2y_i(F_{m-1}(\mathbf{x}_i) + \gamma))).$$

这个值不好算，可以使用Newton-Raphson近似：

$$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i \Big/ \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|)$$

整个过程就是：

ALGORITHM 5 (L_K_TreeBoost).

$F_0(\mathbf{x}) = \frac{1}{2}\log\frac{1+\bar{y}}{1-\bar{y}}$

For $m = 1$ to M do:

$\tilde{y}_i = 2y_i/(1 + \exp(2y_i F_{m-1}(\mathbf{x}_i))), \; i = 1, N$

$\{R_{jm}\}_1^J = J$-terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$

$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|), \; j = 1, J$

$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$

endFor

end Algorithm

第一行，由于损失函数的不同，初始化与之前的回归不一样。

第二行同样是M次迭代。

第三行求负梯度。

第四行和回归的做法一样，一般用平方误差拟合出一颗回归树。

第五行利用损失函数求叶子节点预测值。

第六行更新强学习器。

在Scikit-learn的代码实现中，会人为在弱学习器前面加个学习率，防止过拟合。

4.2 多类逻辑回归和分类

多类逻辑回归和二类逻辑回归类似，但有一点很不同，K类逻辑回归训练K个强学习器：

ALGORITHM 6 (L_K_TreeBoost).

$F_{k0}(\mathbf{x}) = 0,\ k = 1, K$

For $m = 1$ to M do:

$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x}))/\sum_{l=1}^{K}\exp(F_l(\mathbf{x})),\ k = 1, K$

For $k = 1$ to K do:

$\tilde{y}_{ik} = y_{ik} - p_k(\mathbf{x}_i),\ i = 1, N$

$\{R_{jkm}\}_{j=1}^{J} = J$-terminal node $tree(\{\tilde{y}_{ik}, \mathbf{x}_i\}_1^N)$

$\gamma_{jkm} = \frac{K-1}{K}\frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)},\ j = 1, J$

$F_{km}(\mathbf{x}) = F_{k,m-1}(\mathbf{x}) + \sum_{j=1}^{J}\gamma_{jkm}1(\mathbf{x} \in R_{jkm})$

endFor

endFor

end Algorithm

K类训练K个分类器，这是多分类任务中的OVR(one vs rest)的策略，不再多说。如果对K类回归做M次迭代，总共要学习M*K个弱学习器，得到K个强学习器。相比二分类，多分类任务使用的损失函数更加复杂：

$$L(\{y_k, F_k(\mathbf{x})\}_1^K) = -\sum_{k=1}^{K} y_k \log p_k(\mathbf{x}),$$

$$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) \Big/ \sum_{l=1}^{K}\exp(F_l(\mathbf{x})).$$

其负梯度为：

$$\tilde{y}_{ik} = -\left[\frac{\partial L(\{y_{il}, F_l(\mathbf{x}_i)\}_{l=1}^K)}{\partial F_k(\mathbf{x}_i)}\right]_{\{F_l(\mathbf{x})=F_{l,m-1}(\mathbf{x})\}_1^K} = y_{ik} - p_{k,m-1}(\mathbf{x}_i),$$

各类叶子区域预测值为：

$$\{\gamma_{jkm}\} = \arg\min_{\{\gamma_{jk}\}}\sum_{i=1}^{N}\sum_{k=1}^{K}\phi\left(y_{ik}, F_{k,m-1}(\mathbf{x}_i) + \sum_{j=1}^{J}\gamma_{jk}1(\mathbf{x}_i \in R_{jm})\right),$$

当然，计算比较复杂。同样使用Newton-Raphson近似：

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)}.$$

5. 反思总结

5.1 样本权重调整

在GBDT中，每一轮基学习器的都是在上一轮的基学习器基础上进行。在新学习器中，已经预测正确的样本的损失为0，在这一轮中不会受到关心，而更关注那些分类错误的样本。因此，GBDT虽然没有像Adaboost那样直接调整样本权重，但也变相增大了错误样本的权重。

5.2 GBDT优缺点

GBDT**优点多多**，比如：

- 弱分类器要求不高，树的层数一般较小，小数据可用，扩展到大数据也能方便处理。
- 需要更少的特征工程，比如不用做特征标准化
- 可以处理字段缺失的数据
- 可以自动组合多个特征并且不用关心特征间是否依赖，可以自动处理特征间的交互，不用担心数据是否线性可分
- 可以灵活处理多种类型的异构数据，这是决策树的天然特性
- 损失函数选择灵活，可以选择具有鲁棒性的损失函数，对异常值有一定的鲁棒性

也有一些**局限性**，比如：

- GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络[13]。
- 训练过程基学习器需要串行训练，只能通过局部并行提高速度。

6. GBDT资料推荐

- 网上有几篇讲得不错，比如**GBDT三部曲**：
  https://blog.csdn.net/qq22238533/article/details/79185969
  https://blog.csdn.net/qq22238533/article/details/79192579
  https://blog.csdn.net/qq_22238533/article/details/79199605

- 还有这个**paper reading**:
  http://xtf615.com/paper/GBM.html
  作者是中科院的薛大佬，GBDT的一些疑问就是这位大佬给我解答的。随便翻了下，发现他还有几篇不错的推荐系统文章:
  http://xtf615.com/categories/%E6%8E%A8%E8%8D%90%E7%B3%BB%E7%BB%9F/

- 另外推荐一个**wepon**大神的讲解:
  http://222.199.222.10/cache/5/03/wepon.me/5aa84bcab4e621a09cc475c348590c35/gbdt.pdf

- 当然，最重要的，是**论文**:
  https://projecteuclid.org/download/pdf_1/euclid.aos/1013203451

参考

[1] https://www.knowledgedict.com/tutorial/ml-gbdt.html
[2] https://zhuanlan.zhihu.com/p/59631419
[3]http://www.mayexia.com/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/GBDT%E7%AE%97%E6%B3%95%E5%8E%9F%E7%90%86%E5%88%86%E6%9E%90/
[4] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of statistics, 1189-1232.
[5] http://wepon.me/files/gbdt.pdf
[6] https://www.cnblogs.com/bnuvincent/p/9693190.html
[7] http://xtf615.com/paper/GBM.html
[8] https://www.zybuluo.com/Dounm/note/1031900#412-%E8%B4%9F%E6%A2%AF%E5%BA%A6%E7%9A%84%E7%90%86%E8%AE%BA%E6%94%AF%E6%92%91
[9] https://blog.csdn.net/zpalyq110/article/details/79527653