GBM算法原理

Boosting:

迭代的过程，每次训练改进上一次的结果

Adaboost:

对训练样本进行加权，每一个弱模型结束后，增加错分样本的权重，减少正确样本的权重

GBM：

每一次弱模型的训练目标是在**之前的模型**的**损失函数**的**负梯度方向**下降

加法模型

前向分步算法

GBDT提升树算法

加法模型

提升树利用加法模型与前向分步算法实现学习的优化过程

考虑加法模型(additive model)

$$f(x)=\sum_{m=1}^{M}\beta_m b(x,\gamma_m)$$

其中，$b(x,\lambda_m)$是基函数，λ_m为基函数的参数，β_m为基函数的系数，显然，该式是一个加法模型

$$\min_{\beta_m,\lambda_m}\sum_{i=1}^{N}L(y_i,\sum_{m=1}^{M}\beta_m b(x_i,\gamma_m))$$

通常这是一个复杂的优化问题，前向分步算法(forward stagewise alogrithm)求解这一优化问题的想法是：因为学习的是加法模型，如果能够从前向后，每一步只学习一个基函数及其系数，逐步逼近以上优化目标函数，那就可以简化器优化复杂度。具体地，每步只需优化如下损失函数：

$$\min_{\beta,\lambda}\sum_{i=1}^{N}L(y_i,\beta b(x_i,\gamma_m))$$

前向分步算法：

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),\dots,(x_N,y_N\}$ 损失函数$L(y,f(x))$,基函数集$\{b(x,\gamma\}$

输出：加法模型$f(x)$

(1) 初始化$f_0(x)=0$

(2) 对m=1,2,...,M

a. 极小化损失函数

$$(\beta_m, \gamma_m) = arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i, \gamma))$$

得到参数β_m, γ_m

b. 更新

$$f_m(x) = f_{m-1}(x) + \beta_m b(x, \gamma_m)$$

(3) 得到加法模型

$$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x, \gamma_m)$$

原来接触的算法：

1. 逻辑回归：

$$z = w^T x + b$$
$$\sigma = \frac{1}{1 + e^{-z}}$$
$$L = -\sum_{i=1}^{N} y_i log\sigma_i + (1 - y_i) log(1 - \sigma_i)$$

此时L使用的logistic处理过后的交叉熵，每一次更新的过程就是可以理解为一次弱模型的训练过程

$$w = w + \alpha^T * \bigtriangledown L(w)$$
$$b = b + \alpha^T * \bigtriangledown L(b)$$

每一次都是在现有的模型基础上更新参数，以达到L的极小值(如果是凸函数，此时就是最小值)

2. 回归（均方误差和绝对误差）：

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

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

GBDT中的梯度算法：

假设：每个弱分类器表示为$h_t(x)$

由$h_1(x), h_2(x), \ldots, h_{t-1}(x)$，串行所得的弱分类器是$f_{t-1}(x)$,其损失函数为$L(y, f_{t-1}(x))$

则本轮的目标是找到一个$h_t(x)$，使得下面的式子最小

$$L(y, f_t(x) = L(y, f_{t-1}(x) + h_t(x))$$

也就是说，本轮需要找到决策树，让样本的损失尽量变得更小。

GBDT的思想：（损失函数：绝对误差）

假如有个人30岁，我们首先用20岁去拟合，发现损失有10岁；

这时我们用6岁去拟合剩下的损失，发现差距还有4岁；

第三轮我们用3岁拟合剩下的差距，差距就只有一岁了；

如果我们的迭代轮数还没有完，可以继续迭代下去，每一轮迭代，拟合的岁数误差都会减小。

最终我们预测的结果：20(第一次拟合结果)+6+3+...≈30

在上面的方法中，损失函数(绝对误差)是可导的，针对这些问题，学者Freidman 提出了使用损失函数的负梯度来拟合本轮损失的近似值

$$r_{m,i} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$$

我们可以使用一棵CART回归树来拟合此时的负梯度方向。**损失函数以绝对误差为例。**

此时，假设第m棵CART树具有J个叶子节点，对应的叶节点区域$R_{mj}, j = 1, 2, \ldots, J$

针对每个叶子节点里的样本，我们求出使损失函数最小，也就是拟合叶子节点最好的输出值c_{mj}

$$c_{mj} = \underbrace{arg\min}_{c} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$$

这样我们就得到了本轮的决策树拟合函数如下：

$$h_m(x) = \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

本轮最终得到的强学习器的表达如下：

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

通过拟合损失函数的负梯度，我们找到了一种通用的拟合损失误差的方法，所以无论是分类问题还是回归问题，我们都可以拟合。区别仅仅在于不同的损失函数不同的负梯度(对于CART，需要拟合的负梯度发生变化)

GBDT常用的损失函数：

着重了解对于回归算法，常用的损失函数有4种：

1. 均方误差

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

2. 绝对误差

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

其对应的负梯度为：

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

3. Huber误差

   它是均方误差和绝对误差的折中产物，对于远离中心的异常点，采用绝对误差；而中心附近的点采用均方误差。这个界限一般用分为数点度量。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：

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

4. 分位数误差

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

   其中θ为分位数，需要我们在回归前提前指定。对应的负梯度为：

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

   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。

XGboost的改进：

最重要的两点：

不只有一阶展开还进行了二阶展开

直方图做差加速

https://github.com/dmlc/xgboost

lightBoost的改进：

https://github.com/Microsoft/LightGBM/tree/master/python-package 官方的python封装

https://github.com/ArdalanM/pyLightGBM 非官方的python封装

还有人专门写了代码PK

https://github.com/tks0123456789/XGBoost_vs_LightGBM

直方图做差

分布式计算