GBDT详解上: 理论

Yafei Zhang(kimmyzhang@tencent.com)

August 4, 2016

1 前言

GBDT(Gradient Boosted Decision Tree)是老牌的ensemble模型, 在10年前那个属于ensemble的年代, 它既在学术界掀起了研究热点, 也在工业界, 甚至比赛界:), 都有广泛的应用.

本系列分上下两篇, 第一篇着重GBDT的理论, 第二篇分析它的著名实现xgboost, 来分析一下它高效的背后暗藏什么玄机.

本文的读者应具有一定的机器学习基础, 尤其是决策树.

2 符号定义

在介绍GBDT之前, 首先规范一下符号, 这些符号在全文中都适用.

f, 决策树. 从数学上讲, 决策树是一个分段函数, 所以它的参数描述了分段方法. 我们用$\{R_j\}_1^J$和$\{b_j\}_1^J$表示决策树的参数, 前者是分段空间(决策树划分出的disjoint 空间), 后者是这些空间上f 输出的函数值. 其中J是叶子节点的数量.

$$f(x;\{R_j,b_j\}_1^J)=\sum_{j=1}^{J}b_j\mathbf{1}(x\in R_j) \tag{1}$$

下文经常用$f(x)$来省略表示$f(x;\{R_j,b_j\}_1^J)$.

在Boosting框架中, f理论上可以是很多weak learner(除了线性函数, 聪明的读者想想是为什么), 在Gradient Boosting框架里, 常用且被证明有效的除了决策树[Friedman 1999a]外, 还有逻辑回归[Friedman 2000]. xgboost中的gbtree, gblinear就对应了这两种实现. 本文将讨论范围限定在决策树上, 不关注其它weak learner.

F, 决策树ensemble. 定义为:

$$F=\sum_{i=0}^{K} f_i \tag{2}$$

f_0是模型初始值, 通常是按照一定原则计算出的常数. 同时定义$F_k=\sum_{i=0}^{k} f_i$.

$\mathcal{D}=\{(x_i,y_i)\}_1^N$, 训练样本.

$\mathcal{L}$, 目标函数. 定义为:

$$\mathcal{L}=\mathcal{L}(\{y_i,F(x_i)\}_1^N)=\underbrace{\sum_{i=1}^{N} L(y_i,F(x_i))}_{\text{Training loss}}+\underbrace{\sum_{k=1}^{K}\Omega(f_k)}_{\text{Regularization}} \tag{3}$$

第一项L是针对样本的Loss. L可以有多种选择: 绝对值误差, 平方误差, logistic loss等.

第二项Ω是正则化函数, 它惩罚f_k的复杂度, 树结构越复杂它的值越大. [Friedman 1999a]中, 并没有用Ω来做正则化, 也就是说$\Omega=0$. [Johnson 2014]和[Chen 2016]选择了简单高效的Ω, 两者大同小异, 下文会介绍后者. 事实证明, Ω对提升效果非常重要.

3 算法推导

首先给出原始的GBDT算法框架.

输入: $\{(x_i, y_i)\}_1^N$, K, L, $\cdots$

1. 初始化f_0

for $k = 1$ **to** K **do**

2.1. $\tilde{y}_i = -\frac{\partial \mathcal{L}(y_i, F_{k-1}(x_i))}{\partial F_{k-1}}$, $i = 1, 2, \cdots, N$

2.2. $\{R_j, b_j\}_1^{J^*} = \arg\min_{\{R_j, b_j\}_1^J} \sum_{i=1}^N \left[\tilde{y}_i - f_k(x_i; \{R_j, b_j\}_1^J)\right]^2$

2.3. $\rho^* = \arg\min_\rho \mathcal{L}(\{y_i, F_{k-1}(x_i) + \rho f_k(x_i)\}_1^N)$

$= \arg\min_\rho \sum_{i=1}^N L(y_i, F_{k-1}(x_i) + \rho f_k(x_i)) + \Omega(f_k)$

2.4. 令$f_k = \rho^* f_k$, $F_k = F_{k-1} + f_k$

end

输出: F_K

Algorithm 1: GBDT算法

乍一看算法1, 熟悉的朋友可能一眼就发现了它和梯度下降十分相似. 没错, Gradient Boosting就是在函数空间的梯度下降. 我们不断减去$\frac{\partial f(x)}{\partial x}$, 可以得到$\min_x f(x)$; 同理不断减去$\frac{\partial \mathcal{L}}{\partial F}$, 就能得到$\min_F \mathcal{L}(F)$.

下面一一解读上面的几个重要步骤.

1, 初始化f_0, 常见方法有:

a)随机初始化;

b)用训练样本中的充分统计量初始化[Friedman 1999a];

c)用其它模型的预测值初始化[Mohan 2011].

GBDT很健壮, 对初始值并不敏感, 但是更好的初始值能够获得更快的收敛速度和质量.

2.1, $\tilde{y}_i$被称作响应(response), 它是一个和残差(residual): $y_i - F_{k-1}(x_i)$正相关的变量, 下文会看到这一点.

2.2, 公式背后表达的是, 使用平方误差训练一颗决策树f_k, 拟合数据$\{(x_i, \tilde{y}_i)\}_1^N$.

2.3, 进行line search. 在2.2中的f是在平方误差下学到的, 这一步进行一次line search, 让f乘以步长ρ后最小化$\mathcal{L}$.

实践中, 当这个最小化问题有解析解时, 直接用解析解带入算法; 否则将$\mathcal{L}(\{y_i, F_{k-1}(x_i) + \rho f_k(x_i)\}_1^N)$在$\rho$处用泰勒公式展开到二阶(这里不是xgboost最早提出的, 就在[Friedman 1999a]中), 然后通过一个Newton Step, 得到泰勒近似后的解析解.

观察2.3步骤中, ρ是乘到了$f_k(x_i)$上, 即等价于把每个叶子节点的值$\{b_j\}_1^J$放大ρ倍. 不妨将2.2和2.3两个过程合并, 直接找出$f_k(x_i)$能够最小化2.3 中的问题. 这时, 就需要将$\mathcal{L}(\{y_i, F_{k-1}(x_i) + f_k(x_i)\}_1^N)$在$f_k$处进行泰勒展开了. 详细的推导放在下面的章节.

2.4, 将训练出来的f_k叠加到F.

总体来说, GBDT就是一个不断拟合响应(残差)并叠加到F上的过程. 在这个过程中, 残差不断变小, Loss不断接近最小值.

4 树生成算法

上文提出了, 算法1的2.2和2.3可以合并.

同时, 给出xgboost中使用的正则化函数:

$$\Omega(f_k) = \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2 \tag{4}$$

记住我们的目标是求f_k, 它最小化目标函数:

$$
\begin{aligned}
\mathcal{L}_k &= \sum_{i=1}^{N} L(y_i, F_{k-1}(x_i) + \rho f_k(x_i)) + \Omega(f_k) \\
&= \sum_{i=1}^{N} L(y_i, F_{k-1} + \rho f_k) + \Omega(f_k) \\
&\approx \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + \underbrace{\frac{\partial L(y_i, F_{k-1})}{\partial F_{k-1}}}_{:=g_i} f_k + \frac{1}{2} \underbrace{\frac{\partial^2 L(y_i, F_{k-1})}{\partial F_{k-1}^2}}_{:=h_i} f_k^2 \right) + \Omega(f_k) \\
&= \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + g_i f_k + \frac{1}{2} h_i f_k^2 \right) + \Omega(f_k) \\
&= \sum_{i=1}^{N} \left(L(y_i, F_{k-1}) + g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2
\end{aligned}
$$

整理出和$\{R_j\}_1^J$, $\{b_j\}_1^J$有关的项:

$$
\begin{aligned}
\mathcal{L}_k(\{b_j\}_1^J, \{R_j\}_1^J) &= \sum_{i=1}^{N} \left(g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2 \\
&= \sum_{x_i \in R_j} \left(g_i \sum_{j=1}^{J} b_j + \frac{1}{2} h_i \sum_{j=1}^{J} b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2 \\
&= \sum_{j=1}^{J} \left(\sum_{x_i \in R_j} g_i b_j + \sum_{x_i \in R_j} \frac{1}{2} h_i b_j^2 \right) + \frac{\gamma}{2} J + \frac{\lambda}{2} \sum_{j=1}^{J} b_j^2 \\
&= \sum_{j=1}^{J} \left(\underbrace{\sum_{x_i \in R_j} g_i}_{:=G_j} b_j + \frac{1}{2} \left(\underbrace{\sum_{x_i \in R_j} h_i}_{:=H_j} + \lambda \right) b_j^2 \right) + \frac{\gamma}{2} J \\
&= \sum_{j=1}^{J} \left(G_j b_j + \frac{1}{2} (H_j + \lambda) b_j^2 \right) + \frac{\gamma}{2} J \qquad (5)
\end{aligned}
$$

现在问题来了: 我们如何同时求得$\{R_j\}_1^J$和$\{b_j\}_1^J$.

为了方便阐述这个问题, 把问题分为两个子问题:

问题1: 如果已经得到了$\{R_j\}_1^J$, 最小化$\mathcal{L}_k$的$\{b_j\}_1^J$是多少?

问题2: 如果将当前节点R_i分裂, 应该在哪一个分裂点使得$\mathcal{L}_k$最小? 这一步我称之为树分裂算法.

上面两个问题的答案归纳的描述出了树生成算法: 对根节点使用树分裂算法, 得到左子树R_L和右子树R_R的同时计算出b_L和b_R. 对每个叶子节点重复上面的分裂过程, 直到满足一定条件后退出.

4.1 问题1

对公式5的b_j求导, 令结果为零, 容易求得:

$$b_j^* = -\frac{G_j}{H_j + \lambda} \tag{6}$$

$j = 1, 2, \cdots, J$.

此时最小的$\mathcal{L}_k$是:

$$\mathcal{L}_k^* = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_j^2}{H_j + \lambda} + \frac{\gamma}{2}J \tag{7}$$

4.2 问题2: 树分裂算法

求$\{R_j\}_1^J$与求$\{b_j\}_1^J$的方法不同, 前者它是对输入x所属空间的一种划分方法, 不连续, 无法求导.

精确得到划分$\{R_j\}_1^J$是一个NP问题, 取而代之, 大家都使用贪心法. 即分裂某节点时, 只考虑对当前节点分裂后, 哪个分裂方案能得到最小的$\mathcal{L}_k$.

像传统决策树一样, CART中的办法也是遍历x的每个维度的每个分裂点, 根据问题1计算$\{b_j\}_1^J$和最小的$\mathcal{L}_k$. 这个过程在xgboost中有所优化, 第二篇会详细介绍.

那么现在把问题2形式化的定义出: 将当前节点R_j分裂成R_L和R_R, 使得分裂后整棵树的$\mathcal{L}_k$最小.

再看看公式7, 整棵树的最小$\mathcal{L}_k$等于每个叶子节点上(最小)Loss的和. 由于整个分裂过程只涉及到3个节点, 其它任何节点的Loss在过程中不变, 这个问题又等价于:

$$min_{R_L,R_R}\frac{G_L^2}{H_L+\gamma}+\frac{G_R^2}{H_R+\gamma}-\frac{(G_L+G_R)^2}{H_L+H_R+\gamma} \tag{8}$$

公式8有明确的物理含义, 前两项分别加上新生成的叶子节点的最小Loss, 第3项是指减去被分裂的叶子节点的最小Loss. 它是将节点R_j分裂成R_L和R_R后, 整棵树最小$\mathcal{L}_k$的降低量, 这个量越大越好. 有点拗口, 慢慢体会.

如果公式8的值小于0(意味$\mathcal{L}_k$不能通过分裂降低), 或者满足其它终止条件, 则不分裂R_j. 否则, 按照最小的$\mathcal{L}_k$计划分裂.

算法2正式的描述了树分裂算法.

算法3描述了使用算法2树分裂算法的改进GBDT算法.

5 LSBoost和LogitBoost

本节挑两个最常用的Loss, 介绍基于这两种Loss的GBDT算法: LSBoost, LogitBoost.

L为平方误差时:

$$L=\frac{(y-F(x))^2}{2} \tag{9}$$

对F求导得到响应(此时响应=残差):

$$\tilde{y}_i=-g_i=-\frac{\partial L(y_i,F(x_i))}{\partial F}=y-F(x) \tag{10}$$

二阶导是:

$$h_i=1 \tag{11}$$

输入: R_j, 落在R_j的训练样本$\{(x_i^{(j)}, y_i^{(j)})\}_1^N$, $\cdots$, 其中$x_i^{(j)} \in \mathbb{R}^m$

decr $= 0$

$G = \sum_{i=1}^N g_i$, $H = \sum_{i=1}^N h_i$

for $k = 1$ **to** m **do**

  $G_L = 0$, $H_L = 0$

  for l *in uniq(sorted(*$\{x_{ik}^{(j)}\}$*))* **do**

    $G_L = G_L + g_l$, $H_L = H_L + h_l$

    $G_R = G - G_L$, $H_R = H - H_L$

    $\text{decr} = \max\left(\text{decr}, \frac{G_L^2}{H_L+\gamma} + \frac{G_R^2}{H_R+\gamma} - \frac{(G_L+G_R)^2}{H_L+H_R+\gamma}\right)$

  end

end

if *decr* < 0 *or* 满足终止条件 **then**

  输出: 保持R_j不分裂

end

输出: 按照最大decr的方案将R_j分裂成R_L和R_R

Algorithm 2: 树分裂算法

输入: $\{(x_i, y_i)\}_1^N$, K, $\cdots$

1. 初始化f_0

for $k = 1$ **to** K **do**

  2.1. 根据公式10或13计算$\tilde{y}_i$, $i = 1, 2, \cdots, N$

  2.2. 由树生成算法得到$\{R_j\}_1^J$和$\{b_j\}_1^J$, g_j和h_j根据具体的L计算.

  2.3. 令$F_k = F_{k-1} + f_k$

end

Algorithm 3: GBDT算法-改进

另外一个是Logistic Loss, 即logistic regression中使用的交叉熵Loss. 顾名思义, 使用这个Loss的时候, 只能用于分类问题.

规定$y \in \{-1, 1\}$, 则

$$L = \log(1 + \exp(-2yF(x))) \tag{12}$$

对F求导得到响应:

$$\tilde{y}_i = -g_i = \frac{2y_i}{1 + \exp(2y_i F(x_i))} \tag{13}$$

二阶导是:

$$h_i = \frac{4\exp(2y_i F(x_i))}{(1 + \exp(2y_i F(x_i)))^2} = |g_i|(2 - |g_i|) \tag{14}$$

将上面两种Loss的$L, \tilde{y}, g_i, h_i$分别带入算法1, 即得到LSBoost和LogitBoost[Friedman 1999a].

输入: $\{(x_i, y_i)\}_1^N$, K, $\cdots$

1. 初始化f_0

for $k = 1$ **to** K **do**

2.1. 根据公式10或13计算$\tilde{y}_i$, $i = 1, 2, \cdots, N$

2.2. 由树生成算法得到$\{R_j\}_1^J$和$\{b_j\}_1^J$, g_j和h_j分别使用公式1011或1314计算.

2.3. 令$F_k = F_{k-1} + f_k$

end

输出: F_K

Algorithm 4: LSBoost & LogitBoost

6 正则化

GBDT有非常快降低Loss的能力, 这也会造成一个问题: Loss迅速下降, 模型低bias, 高variance, 造成过拟合.

下面一一介绍GBDT中抵抗过拟合的技巧.

限制树的复杂度. Ω函数对树的节点数, 和节点上预测值$\{b_j\}_1^J$的平方和均有惩罚. 除此之外, 我们通常在终止条件上还会增加一条: 树的深度.

采样. 即训练每个树的时候, 只使用一部分的样本.

列采样. 即训练每个树的时候, 只使用一部分的特征. 这时xgboost的创新, 它将随机森林中的思想引入了GBDT.

Shrinkage. 进一步惩罚$\{b_j\}_1^J$, 给它们乘一个小于1的系数. 也可以理解为设置了一个较低的学习率.

Early stop. 因为GBDT的可叠加性, 我们使用的模型不一定是最终的ensemble, 而根据测试集的测试情况, 选择使用前若干棵树.

7 思考

请读者思考两个问题:

1. 算法1和算法3的两个方法是否等价? 如何证明等价咧?

2. [Friedman 1999a]的推导和xgboost的推导是否等价?

两个答案都是肯定的, 有兴趣的读者可以自行证明.

8 参考文献

[Friedman 1999a] J. Friedman(1999). Greedy Function Approximation: A Gradient Boosting Machine.

[Friedman 1999b] J. Friedman(1999). Stochastic Gradient Boosting.

[Friedman 2000] J. Friedman, T. Hastie, R. Tibshirani(2000). Additive Logistic Regression: A Statistical View of Boosting.

[Mohan 2011] A. Mohan, Z. Chen, K. Weinberger(2011). Web-Search Ranking with Initialized Gradient Boosted Regression Trees.

[Johnson 2014] R. Johnson, T. Zhang(2014). Learning Nonlinear Functions Using Regularized Greedy Forest

[Chen 2016] T. Chen, C. Guestrin(2016). XGBoost: A Scalable Tree Boosting System.