$GBDT$算法原理梳理

杨航锋

1 $GBDT$ 算法总述

$GBDT$ 的全称为($\boldsymbol{G}radient\ \boldsymbol{B}oosting\ \boldsymbol{D}ecision\ \boldsymbol{T}ree$)，是一种广泛用于分类、回归和推荐系统中排序任务的机器学习算法，属于$Boosting$ 算法族。$Boosting$ 算法的原理是先从初始训练集中训练一个基学习器，然后再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器分类错误的训练样本在后续的训练过程中受到更多关注，然后基于调整后的样本分布来训练下一个基分类器；如此反复进行，直至基学习器数目达到事先指定的值 T ，最后将这 T 个基学习器进行加权结合。值得注意的是 $GBDT$ 中的树都是**回归树**而非分类树，$GBDT$ 的迭代更新过程即不断缩小残差的过程。

2 $Boosting$ 加法模型和前向分布算法

令输入空间是 $X \in \mathbb{R}^n$,输出空间是 $Y \in \mathbb{R}$ ，不妨假设含有m个样本数据($x^{(1)}$,$y^{(1)}$)、($x^{(2)}$,$y^{(2)}$)、$\cdots$、($x^{(m)}$,$y^{(m)}$)，其中$x^{(i)} \in X$、$y^{(i)} \in Y$ 。$GBDT$ 算法的假设函数可以看成是由 T 棵树组合而成的加法模型：

$$\hat{y}^{(i)} = \sum_{t=1}^{T} f_t(x^{(i)}), f_t \in \mathscr{F}$$

其中 $\mathscr{F}$ 为所有树组合而成的函数空间，该模型的待求决策树函数为 $\Theta_T = \{f_1, f_2, \cdots, f_T\}$ ，因此可以定义上述加法模型的目标函数为 $Obj = \sum_{i=1}^{m} l(y^{(i)}, \hat{y}^{(i)})$ ，通过前向分布算法来优化该目标函数：

$$\begin{aligned}
\hat{y}_0^{(i)} &= \epsilon \\
\hat{y}_1^{(i)} &= f_1(x^{(i)}) = \hat{y}_0^{(i)} + f_1(x^{(i)}) \\
\hat{y}_2^{(i)} &= f_1(x^{(i)}) + f_2(x^{(i)}) = \hat{y}_1^{(i)} + f_2(x^{(i)}) \\
&\cdots \\
\hat{y}_k^{(i)} &= \sum_{t=1}^{k} f_t(x^{(i)}) = \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})
\end{aligned}$$

所以，迭代更新到第 k 步时，目标函数可改写为：

$$Obj^{(k)} = \sum_{i=1}^{m} l\left(y^{(i)}, \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})\right)$$

最优化 $Obj^{(k)}$ 即可求得该轮需要学习的决策树函数 $f_k(x^{(i)})$ 。

3 *Gradient Boosting* 算法

Gradient Boosting 以负梯度代替残差来求解 $f_k(x^{(i)})$ ，**对于任意可导函数 $g(x)$ 来说，总是存在如下关系 $g(x - \epsilon \nabla_x g(x)) < g(x)$** ，*Gradient Boosting* 算法利用损失函数的负梯度在当前模型的值作为之前残差的近似值来拟合回归树 $f_k(x^{(i)})$ 。

> **泰勒公式**：设 n 是一个正整数，如果定义在包含 x_0 的区间上的函数 f 在 x_0 点处 $n+1$ 次可导，那么对于这个区间上的任意 x 都有：
>
> $$f(x) = f(x_0) + \frac{f^{(1)}(x_0)}{1!}(x - x_0) + \frac{f^{(2)}(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x)$$
>
> 等号右边的多项式称为函数 $f(x)$ 在 x_0 处的泰勒展开式， $R_n(x)$ 是泰勒公式的余项且是 $(x - x_0)^n$ 的高阶无穷小。

为了求 $l\left(y^{(i)}, \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})\right)$ 在 $\hat{y}_{k-1}^{(i)}$ 处的一阶展开，不妨先对 $l(y^{(i)}, x)$ 在 $\hat{y}_{k-1}^{(i)}$ 处进行一阶展开可得：

$$l(y^{(i)}, x) \simeq l(y^{(i)}, \hat{y}_{k-1}^{(i)}) + \nabla_{\hat{y}_{k-1}^{(i)}} l(y^{(i)}, \hat{y}_{k-1}^{(i)}) \cdot (x - \hat{y}_{k-1}^{(i)})$$

令 $x = \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})$ ，且记 $\nabla_{\hat{y}_{k-1}^{(i)}} l\left(y^{(i)}, \hat{y}_{k-1}^{(i)}\right)$ 为 g_i 则有：

$$l\left(y^{(i)}, \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})\right) \simeq l(y^{(i)}, \hat{y}_{k-1}^{(i)}) + g_i f_k(x^{(i)})$$

故目标函数最终形式为：

$$Obj^{(k)} = \sum_{i=1}^{m} \left(l(y^{(i)}, \hat{y}_{k-1}^{(i)}) + g_i f_k(x^{(i)})\right)$$

4 *XGBoost* 算法

XGBoost 算法是 *GBDT* 算法的改进版本，其目标函数为：

$$\begin{aligned} Obj^{(k)} &= \sum_{i=1}^{m} l(y^{(i)}, \hat{y}_k^{(i)}) + \sum_{i=1}^{T} \Omega(f_i) \\ &= \sum_{i=1}^{m} l(y^{(i)}, \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})) + \Omega(f_k) + C \end{aligned}$$

同理为了求损失函数 $l\left(y^{(i)}, \hat{y}_{k-1}^{(i)}+f_k(x^{(i)})\right)$ 在 $\hat{y}_{k-1}^{(i)}$ 处的二阶展开，不妨先对 $l(y^{(i)}, x)$ 在 $\hat{y}_{k-1}^{(i)}$ 处进行二阶展开可得：

$$l(y^{(i)}, x) \simeq l(y^{(i)}, \hat{y}_{k-1}^{(i)}) + \nabla_{\hat{y}_{k-1}^{(i)}} l(y^{(i)}, \hat{y}_{k-1}^{(i)}) \cdot (x - \hat{y}_{k-1}^{(i)}) + \frac{1}{2}\nabla^2_{\hat{y}_{k-1}^{(i)}} l(y^{(i)}, \hat{y}_{k-1}^{(i)}) \cdot (x - \hat{y}_{k-1}^{(i)})^2$$

令 $x = \hat{y}_{k-1}^{(i)} + f_k(x^{(i)})$，且记 $\nabla_{\hat{y}_{k-1}^{(i)}} l\left(y^{(i)}, \hat{y}_{k-1}^{(i)}\right)$ 为 g_i 、 $\nabla^2_{\hat{y}_{k-1}^{(i)}} l\left(y^{(i)}, \hat{y}_{k-1}^{(i)}\right)$ 为 h_i 则有：

$$l\left(y^{(i)}, \hat{y}_{k-1}^{(i)}+f_k(x^{(i)})\right) \simeq l(y^{(i)}, \hat{y}_{k-1}^{(i)}) + g_i f_k(x^{(i)}) + \frac{1}{2} h_i f_k^2(x^{(i)})$$

又因为在第 k 步 $\hat{y}_{k-1}^{(i)}$ 其实是已知的，所以 $l(y^{(i)}, \hat{y}_{k-1}^{(i)})$ 是一个常数函数，故对优化目标函数不会产生影响，将上述结论带入目标函数 $Obj^{(k)}$ 可得：

$$Obj^{(k)} \simeq \sum_{i=1}^{m}\left[g_i f_k(x^{(i)}) + \frac{1}{2} h_i f_k^2(x^{(i)})\right] + \Omega(f_k)$$

4.1 优化目标函数

以 $XGBoost$ 算法的目标函数为例，对于任意决策树 f_k ，假设其叶子结点个数 T ，该决策树是由所有结点对应的值组成的向量 $w \in \mathbb{R}^T$ ，以及能够把特征向量映射到叶子结点的函数 $q(*) : \mathbb{R}^d \to \{1, 2, \cdots, T\}$ 构造而成的，且每个样本数据都存在唯一的叶子结点上。因此决策树 f_k 可以定义为 $f_k(x) = w_{q(x)}$ 。决策树的复杂度可以由正则项 $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$ 来定义，该正则项表明决策树模型的复杂度可以由叶子结点的数量和叶子结点对应值向量 w 的 $L2$ 范数决定。定义集合 $I_j = \{i | q(x^{(i)}) = j\}$ 为划分到叶子结点 j 的所有训练样本的集合，即之前训练样本的集合，现在都改写成叶子结点的集合，因此 $XGBoost$ 算法的目标函数可以改写为：

$$\begin{aligned}
Obj^{(k)} &\simeq \sum_{i=1}^{m}\left[g_i f_k(x^{(i)}) + \frac{1}{2} h_i f_k^2(x^{(i)})\right] + \Omega(f_k) \\
&= \sum_{i=1}^{m}\left[g_i w_{q(x^{(i)})} + \frac{1}{2} h_j w_{q(x^{(i)})}^2\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T}\left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2\right] + \gamma T
\end{aligned}$$

令 $G_j = \sum\limits_{i \in I_j} g_i$, $H_j = \sum\limits_{i \in I_j} h_i$ 则有：

$$Obj^{(k)} \simeq \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j+\lambda)w_j^2\right]$$

分析可知当更新到第 k 步时，此时**决策树结构固定的情况下**，每个叶子结点有哪些样本是已知的，那么 $q(*)$ 和 I_j 也是已知的；又因为 g_i 和 h_i 是第 $k-1$ 步的导数，那么也是已知的，因此 G_j 和 H_j 都是已知的。令目标函数 $Obj^{(k)}$ 的一阶导数为 0，即可求得叶子结点 j 对应的值为：

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

因此针对于结构固定的决策树，最优的目标函数 Obj 为：

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

上面的推导是建立在决策树结构固定的情况下，然而决策树结构数量是无穷的，所以实际上并不能穷举所有可能的决策树结构，什么样的决策树结构是最优的呢？通常使用贪心策略来生成决策树的每个结点，$XGBoost$ 算法的在决策树的生成阶段就对过拟合的问题进行了处理，因此无需独立的剪枝阶段，具体步骤可以归纳为：

1. 从深度为 0 的树开始对每个叶子结点穷举所有的可用特征；
2. 针对每一个特征，把属于该结点的训练样本的该特征升序排列，通过线性扫描的方式来决定该特征的**最佳分裂点**，并采用最佳分裂点时的**收益**；
3. 选择收益最大的特征作为分裂特征，用该特征的最佳分裂点作为分裂位置，把该结点生成出左右两个新的叶子结点，并为每个新结点关联新的样本集；
4. 退回到第一步，继续递归操作直到满足特定条件。

因为对某个结点采取的是二分策略，分别对应左子结点和右子结点，除了当前待处理的结点，其他结点对应的 Obj 值都不变，所以对于收益的计算只需要考虑当前结点的 Obj 值即可，分裂前针对该结点的最优目标函数为：

$$Obj^{(before)} = -\frac{1}{2}\frac{(G_L+G_R)^2}{(H_L+H_R)+\lambda} + \gamma$$

分裂后的最优目标函数为：

$$Obj^{(later)} = -\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda}\right] + 2\gamma$$

那么对于该目标函数来说，分裂后的收益为：

$$
\begin{aligned}
Gain &= Obj^{(before)} - Obj^{(later)} \\
&= \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{(H_L+H_R)+\lambda}\right]-\gamma
\end{aligned}
$$

故可以用上述公式来决定最有分裂特征和最优特征分裂点。

4.2 总结

$XGBoost$ 算法的过程可以归纳为：

1. 前向分布算法的每一步都生成一棵决策树；
2. 拟合该决策树之前，先计算损失函数在每个样本数据上的一阶导 g_i 和二阶导 h_i ；
3. 通过贪心策略生成一棵决策树，计算每个叶子结点的 G_j 和 H_j 并计算预测值 w ；
4. 把新生成的决策树 $f_k(x)$ 加入 $\hat{y}_k^{(i)}=\hat{y}_{k-1}^{(i)}+\epsilon f_k(x^{(i)})$ ，其中 ϵ 是学习率主要控制模型的过拟合。

5 $XGBoost$ 的优势

相比于普通的 $GBDT$ 算法 $XGBoost$ 算法的主要优点在于：

- 不仅支持决策树作为基分类器，还支持其它线性分类器；
- 使用了损失函数的二阶泰勒展开，因此与损失函数更接近，收敛速度更快；
- 在目标函数中加入了正则项，用于控制模型的复杂度；
- $Shrinkage$ 也就是之前说的 ϵ ，主要用于削弱每棵决策树的影响，让后面有更大的学习空间，实际应用中一般把 ϵ 设置的小点，迭代次数设置的大点；
- 列抽样， $XGBoost$ 从随机森林算法中借鉴而来，支持列抽样可以降低过拟合，并且减少计算；
- 支持对缺失值的处理，对于特征值缺失的样本， $XGBoost$ 可以学习这些缺失值的分裂方向；
- 支持并行，在对每棵决策树进行结点分裂时，需要每个特征的增益，选择最大的那个特征作为分裂特征，各个特征的增益计算可以多线程进行；
- 近似算法，决策树结点在分裂时需要穷举每个可能的分裂点，当数据没法全部加载到内存中时，这种方法会比较慢， $XGBoost$ 提出了一种近似的方法去高效的生成候选分割点。