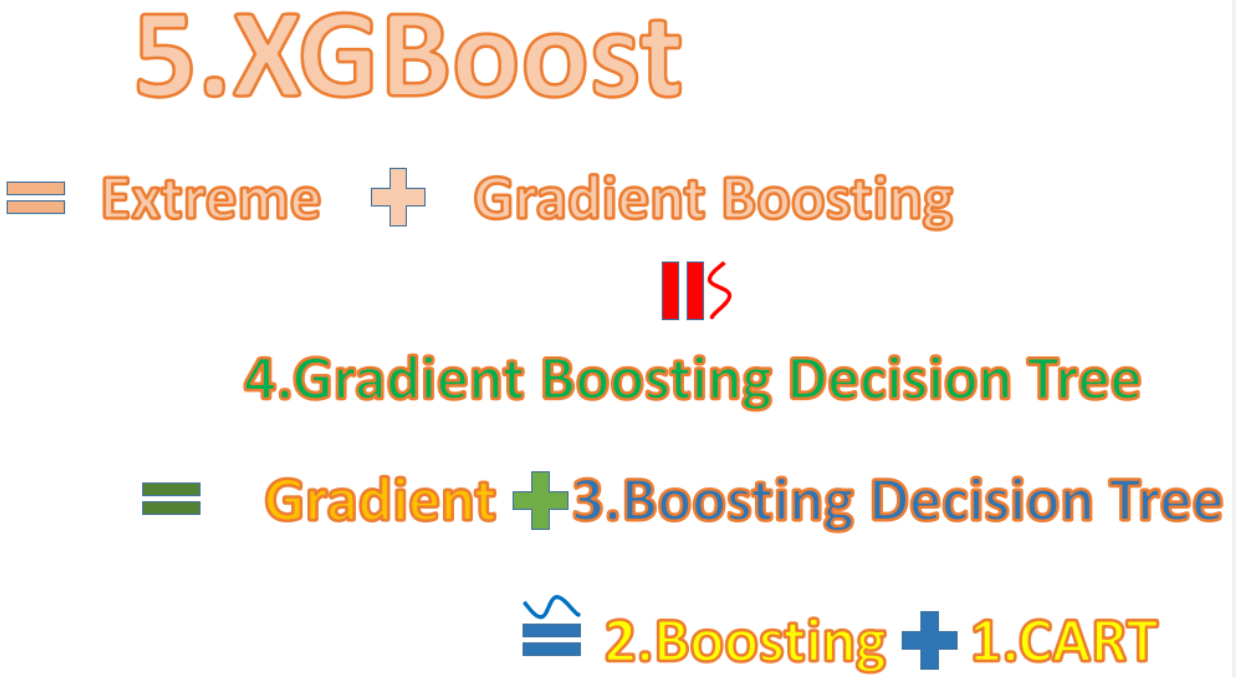
Table of Contents

XGBoost

简介

在大数据竞赛中,XGBoost霸占了文本图像等领域外几乎80%以上的大数据竞赛.当然不仅是在竞赛圈,很多大公司也都将XGBoost作为核心模块使用,好奇的人肯定都很想揭开这个神奇的盒子的幕布,究竟是里面是什么,为什么这么厉害? 本篇notebook会从理论和实践的角度来讲述XGBoost以及关于它的一段历史与组成.

此处我们会按照下图的形式来讲述关于XGBoost的进化史.



CART(Classification and Regression Tree)

CART的全称是Classification and Regression Tree,翻译过来就是分类与回归树,是由四人帮Leo Breiman, Jerome Friedman, Richard Olshen与Charles Stone于1984年提出的,该算法是机器学习领域一个较大的突破,从名字看就知道其既可用于分类也可用于回归.

CART本质是对特征空间进行二元划分(即CART生成的决策树是一棵二叉树),它能够对类别变量与连续变量进行分裂,大体的分割思路是**先对某一维数据进行排序(这也是为什么我们对无序的类别变量进行编码的原因[1]),然后对已经排好后的特征进行切分,切分的方法就是if ... else ...的格式.然后计算衡量指标(分类树用Gini指数,回归树用最小平方值),最终通过指标的计算确定最后的划分点[2],然后按照下面的规则生成左右子树:**

> If x < A: Then go to left; else: go to right.

分裂指标（分类树：Gini指数）

- **对于给定的样本集D的Gini指数: 分类问题中,假设有K个类,样本点属于第k类的概率为p_k,则概率分布的基尼指数为:$Gini(D) = \Sigma_{k=1}^{K} p_k(1-p_k)$[3]**.

> ①.从Gini指数的数学形式中,我们可以很容易的发现,当$p_1 = p_2 = ... = p_K$的时候**Gini指数是最大**的,这个时候分到每个类的概率是一样的,判别性极低,对我们分类带来的帮助很小,可以忽略.
> ②.当某些p_i较大,即第i类的概率较大,此时我们的Gini指数会变小意味着判别性较高.样本中的类别不平衡.

- **在给定特征A的条件下,样本集合D的基尼指数(在CART中)为:**
  $Gini(D, A) = p_1 Gini(D_1) + p_2 Gini(D_2)$,**其中** $p_i = \frac{D_i}{D_1 + D_2}, i \in 1, 2$

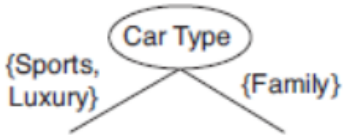
在CART分割时,我们**按照Gini指数最小来确定分割点的位置[4]**.

例子

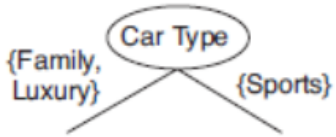
对于Categorical变量

数据集关于第一个特征维度的Gini指数的计算(其他的类似)就是:

$$Gini(D, CarType) = \frac{9+7}{9+7+1+3}[2 * \frac{9}{9+7} * (1 - \frac{9}{9+7})] + \frac{1+3}{9+7+1+3}[2 * \frac{1}{1+3} * (1 - \frac{1}{1+3})] = 0.46$$

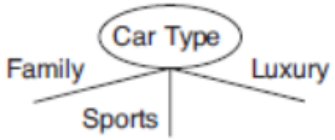


Car Type		
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	



Car Type		
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

(a) Binary split



Car Type			
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(b) Multiway split

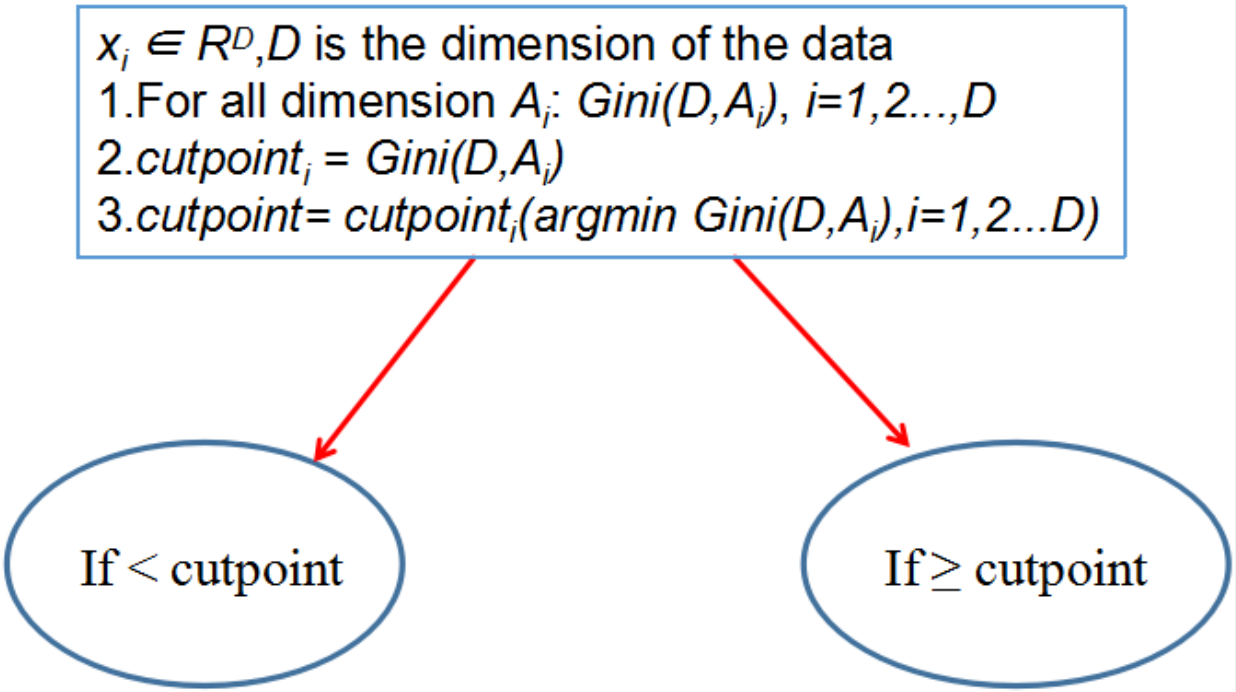
对于连续变量

因为连续变量的部分已经通过某一个值进行了划分,所以计算的时候和Categorical变量的计算类似.

Class	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual Income										
Sorted Values →	60	70	75	85	90	95	100	120	125	220
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	*0.300*	0.343	0.375	0.400	0.420

CART分割示意图

下图的$Gini(D, A_i)$默认为第i个特征维度的最小Gini值.



CART算法(分类树,无剪枝)

输入:训练数据集D,停止计算条件.
输出:CART决策树.

根据训练数据集,从根节点开始,递归地对每个结点进行以下操作,构建二叉决策树:
(1) 设结点的训练数据集为D,计算现有特征对该数据集的基尼指数(Gini),此时,对每一个特征A,对其可能取的**每个值**a**(先排序[5])**,根据样本点对$A = a$的测试"是"或"否"将D分割成D_1,D_2,计算当$A = a$时的基尼指数;

(2) 在所有可能的特征A以及它们所有可能的切分点a中,选择基尼指数最小的特征及其对应的切分点作为最优特征与最优切分点,依最优特征与最优切分点,从现节点生成两个子节点,将训练数据集依特征分配到两个子节点中去;
(3) 对两个子节点递归调用(1),(2),直至满足停止条件.
(4) 生成CART决策树.

CART（回归树部分）

输入:训练数据集D,停止计算条件.
输出:CART回归树$f(x)$.

在训练数据集所在的输入空间中,递归地将每个区域划分为两个子区域并决定每个子区域的输出值,构建二叉决策树:
(1) 选择最优切分变量j与切分点s,求解:
$min_{j,s}[min_{c_1}\Sigma_{x_i\in R_1(j,s)}(y_i-c_1)^2+min_{c_2}\Sigma_{x_i\in R_2(j,s)}(y_i-c_2)^2$
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:

$$R_1(j,s)=\{x|x^{(j)}\leq s\},R_2(j,s)=\{x|x^{(j)}>s\}$$
$$\bar{c}_m=\frac{1}{N_m}\Sigma_{x_i\in R_m(j,s)}y_i,x\in R_m,m=1,2$$

(3) 继续对两个子区域调用(1),(2),直至满足停止条件.
(4) 将输入空间划分为M个区域$R_1,R_2,\ldots,R_M$,生成决策树: $f(x)=\Sigma_{m=1}^{M}\bar{c}_mI(x\in R_m)$

附录

- 由上面的分析我们可以知晓,**CART是基于单特征的,没有考虑特征之间的关联性,但这也给了我们两个提示,①我们的特征是不需要进行归一化处理的[6];②有时我们需要通过特征之间的相关性来构建新的特征[7].**.
- 因为本篇notebook的核心是XGBoost,关于CART剪枝以及其他相关的内容就不再累述.

Boosting

上面是最常用的树模型的介绍,现在我们要介绍Boosting技术,Boosting在维基百科上面的解释是:

- **Boosting**: a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones[8].

实际生活中,人们也常用"三个臭皮匠赛过诸葛亮"的话来描述Boosting(提升方法).当然这背后还有理论的支撑与保障,两个大牛Kearns和Valiant提出了强可学习和弱可学习概念同时在此概念下,Schapire证明的**强可学习与弱可学习是等价的伟大理论[9]**.

服从Boosting维基百科的解释,我们用简单的数学对其进行表示,

- **多个臭皮匠 ---- 多个弱分类器($f_1(x), f_2(x), \ldots, f_M(x)$)**

现在我们要将多个弱分类器转化为强分类器$F(x)$.怎么转换呢？最直接的就是线性加权:
$F(x) = \Sigma_{i=1}^{M} \alpha_i f_i(x)$,其中 α_i 为第i个分类器的权重, 当然我们也可以将其进一步变形为:
$F(x) = \Sigma_{i=1}^{M} \alpha_i f_i(x) = \Sigma_{i=1}^{M} g_i(x)$,其中$g_i(x) = \alpha_i f_i(x)$.

Boosting Tree

将上面的Boosting思想和树进行结合,我们便得到了提升树的简易版本.
$F(x) = \Sigma_{i=1}^{M} g_i(x) = \Sigma_{i=1}^{M} g(x, \theta_i)$, $g(x, \theta_i)$ 是第i棵树, θ_i是第i棵树的参数.我们就得到了Boosting Tree的表达形式.但是这样的形式该如何求解得到呢.....

Boosting的基本思想

按照所有的监督机器学习算法的模式,我们需要通过训练获得的$F(x)$,使得对于给定的输入变量x,我们输出的$F(x)$都能较好地近似y.换言之,就是希望我们的损失函数:$L(y, F(x))$尽可能的小.

如果从传统的平方损失函数的角度出发的话就是: 希望对于所有的样本, $min\frac{1}{N}\Sigma_{i=1}^{N}(y_i - F(x_i))^2$,写成向量的形式就是 $min\frac{1}{N}(y - F(x))^2$. 因为$F(x)$是由多个含有不同参数的弱分类器组成,我们无法用传统的梯度下降的形式进行直接的优化, 不过不急, 我们慢慢分析, 一步一步来.

我们的**目标是最小化 $min(y - F(x))^2$, y是向量的形式,包含所有样本的label**.

> ① 我们构造$f_1(x)$希望(y-f_1(x))^2 尽可能的小.
> ② 我们训练$f_2(x)$希望(y-f_1(x)-f_2(x))^2尽可能的小.
> ③ 我们训练$f_3(x)$希望(y-f_1(x)-f_2(x)-f_3(x))^2尽可能的小.
> .....依次类推,直至第$f_M(x)$.

从上面的构造角度看,我们发现构建第$t+1$个分类器的时候,前t个分类器是固定的,也就是说在第$t+1$步,我们的目标就是$min(y - \Sigma_{j=1}^{t} f_j(x) - f_{t+1}(x))^2$ 我们令$r = y - \Sigma_{j=1}^{t} f_j(x)$表示我们的残差,而我们的下一个分类器$f_{t+1}(x)$就是尽可能拟合我们的残差r. 使得
$(y - \Sigma_{j=1}^{t} f_j(x) - f_{t+1}(x))^2 = (r - f_{t+1}(x))^2 < (y - \Sigma_{j=1}^{t} f_j(x))^2$,这样我们每次迭代一轮,损失误差就会继续变小,在训练集上离我们的目标也就更加近了.

从而我们得到我们的梯度提升树算法.

提升树算法 (回归问题)[10]

输入: 训练数据集 $T = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N), x_i \in X, y_i \in Y$;
输出: 提升树$f_M(x)$
(1) 初始化 $f_0(x) = 0$.
(2) 对 $m = 1, 2, \ldots, M$
(a) 计算残差, $r_{mi} = y_i - F_{m-1}(x_i), i = 1, 2, 3, \ldots, N$.
(b) 拟合残差r_{mi}学习一个回归树,得到$f_m(x)$.
(c) 更新$f_m(x) = F_{m-1}(x) + f_m(x)$.
(3) 得到回归问题提升树 $F_M(x) = \Sigma_{i=1}^{M} f_i(x)$

问题

上面我们使用的是简单的平方损失函数的形式对我们的目标进行了展开与化简,但我们**实际中却有很多其他的损失函数,而且在很多问题中,这些损失函数比我们的平方损失函数要好很多. 同样的, 我们的方法仍然不变,是希望在第$t+1$步的时候学习分类器$f_{t+1}(x)$ 使得 $L(y, \Sigma_{j=1}^{t} f(x) + f_{t+1}(x))$ 尽可能的在 $L(y, \Sigma_{j=1}^{t} f(x))$的基础上面变得更小**. 这个时候我们发现无法像上面那样直接展开并进行残差拟合了.....那该怎么办呢？

Gradient Boosting Decision Tree(梯度提升树)

Gradient Boosting的基本思想

接着上面的问题,我们退回来想一想,因为我们的目标是最小化$L(y, F(x))$,所以我们只需要$L(y, \Sigma_{j=1}^{t} f_j(x) + f_{t+1}(x))$的值比$L(y, \Sigma_{j=1}^{t} f_j(x))$小,好了,既然这样,貌似问题又出现了一丝曙光,那现在我们再假设,我们的损失函数L是可导的,这个假设很合理,毕竟现在90%以上的损失函数都是可导的.是不是感觉至少该问题又可以求解了.

那么我们现在的目标就变成了 $max[(L(y, \Sigma_{j=1}^{t} f_j(x)) - L(y, \Sigma_{j=1}^{t} f_j(x) + f_{t+1}(x)))]$,为了方便我们令$c = \Sigma_{j=1}^{t} f_j(x)$, 好吧,既然如此, $max[L(y, c) - L(y, c + f_{t+1}(x))]$

> $L(c + f_{t+1}(x)) \approx L(c) + L'(c) f_{t+1}(x)$,
>
> $L(c + f_{t+1}(x)) = L(c) + L'(c) f_{t+1}(x) = L(c) - L'(c)^2 < L(c)$ 其中 $f_{t+1}(x) = -1 * L'(x)$

$\rightarrow\rightarrow\rightarrow\rightarrow$现在我们用$f_{i+1}(x)$来拟合$f_{t+1}(x) = -1 * L'(x)$, 原先的 r 就变成了现在的梯度！ Nice! 于是梯度提升树就产生了.

梯度提升树算法[11]

输入: 训练数据集 $T = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N), x_i \in X \subset R^n, y_i \in Y \subset R$; 损失函数 L(y,f(x)),树的个数M.
输出: 梯度提升树$F_M(x)$
(1) 初始化 $f_0(x) = argmin_c \Sigma_{i=1}^{N} L(y_i, c)$.
(2) 对 $m = 1, 2, \ldots, M$
(a) 对$i = 1, 2, \ldots, N$,计算, $r_{mi} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=F_{m-1}(x)}$.
(b) 拟合残差r_{mi}学习一个回归树,得到$f_m(x)$.
(c) 更新$F_m(x) = F_{m-1}(x) + f_m(x)$.
(3) 得到回归问题提升树 $F_M(x) = \Sigma_{i=0}^{M} f_i(x)$

注意

- 上面提到的梯度提升树是最简易的版本,后续写在各种开源包中的GBDT都在残差的部分增加了步长(或者学习率), $r_{mi} = -$ $\alpha_m$$[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=F_{m-1}(x)}$, 用学习率来控制我们模型的学习速度,和传统的梯度下降的方式类似.

XGBoost

上面的部分讲述了GBDT的原理部分,现在正式进入主题,神奇的XGBoost,XGBoost展开的意思就是Extreme Gradient Boosting,其中Extreme是极致的意思,主要体现在工程设计层面,包括并发的程序执行贪心的排序操作等,因为本篇notebook的核心不在于此,接下来我们着重介绍XGBoost在实现与原理上于传统GBDT的区别.

XGBoost与传统GBDT的算法层面的区别

XGBoost中的GBDT与传统的GBDT在算法层面有主要有两处较大的不同:

- XGBoost中的导数不是一阶的,是二阶的[12];
- XGBoost中的剪枝部分在对叶子的个数做惩罚的同时还加入权重的惩罚.换言之,正则项进行了改进[13].

XGBoost中的二阶梯度

> $L(x+\delta x) \approx L(x) + L'(x)\delta x + \frac{1}{2}L''(x)\delta x^2$,
>
> $L(x+\delta x) = L(x) + L'(x)\delta x + \frac{1}{2}L''(x)\delta x^2 = L(x) - \frac{1}{2}\frac{L'(x)^2}{L''(x)} < L(x)$ 其中 $\delta x = -1 * \frac{L'(x)}{L''(x)}$

其实这个对应的就是牛顿提升树.

Newton提升树 [12]

输入: 训练数据集 $T = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N), x_i \in X \subset R^n, y_i \in Y \subset R$; 损失函数 L(y,f(x)),树的个数M.
输出: 梯度提升树$F_M(x)$
(1) 初始化 $f_0(x) = argmin_c \Sigma_{i=1}^{N} L(y_i, c)$.
(2) 对 $m = 1, 2, \ldots, M$
(a) 对$i = 1, 2, \ldots, N$,计算, $r_{mi} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} + \frac{1}{2}\frac{\partial_2 L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=F_{m-1}(x)}$.

(b) 拟合残差r_{mi}学习一个回归树,得到$f_m(x)$.
(c) 更新$F_m(x) = F_{m-1}(x) + f_m(x)$.
(3) 得到回归问题提升树 $F_M(x) = \Sigma_{i=1}^{M} f_i(x)$

XGBoost中的正则项

传统的GBDT为了控制树的复杂度常常会对树的叶子个数加正则项进行控制, XGBoost不仅仅对于树中的叶子节点的个数进行控制,与此同时还对每个叶子节点的分数加入正则.即:

- 传统的GBDT的损失函数 $\Sigma_{i=1}^{N} L(y_i, F(x_i)) + \Omega(F)$ 其中 $F(x) = \Sigma_{i=1}^{M} f_i(x)$ 通常

- 传统的GBDT的损失函数: $\Sigma_{i=1}^{N} L(y_i, F(x_i)) + \Omega(F)$,其中$F(x) = \Sigma_{i}^{M} f_i(x)$, 通常$\Omega(F) = \gamma\Sigma_{i=1}^{M} T_i$, T_i 表示第i棵树的叶子节点的个数.
- XGBoost的损失函数: $\Sigma_{i=1}^{N} L(y_i, F(x_i)) + \Omega(F)$,其中$F(x) = \Sigma_{i}^{M} f_i(x)$, 通常$\Omega(F) = \gamma\Sigma_{i=1}^{M} T_i$+$\frac{1}{2}\lambda\Sigma_i w_i^2$, T_i 表示第i棵树的叶子节点的个数, w_i为第i个叶子节点的值.

XGBoost的推导与算法构建

XGBoost的splitting准则的推导

下面我们对XGBoost中的一些公式进行推导,方便后续对于树的splitting有更好的理解.我们的目标依然不变,就是希望构建一棵新的树的时候,我们的损失函数尽可能的变得更小,即:

$min\ \Sigma_{k=1}^{n}(L(y_k, \bar{y}_k + f_{t+1}(x_k)) - L(y_k, \bar{y}_k))$,其中 $\bar{y}_k = \Sigma_{j=1}^{t} f_j(x_k)$, $\bar{y}_k$表示前t棵树对于某个变量x_k的预测值.

使用二阶泰特展开并且去掉常数值,我们可以得到:
$min\ \Sigma_{k=1}^{n}(L(y_k, \bar{y}_k + f_{t+1}(x_k)) - L(y_k, \bar{y}_k)) \approx min\ \Sigma_{k=1}^{n}(g_k f_{t+1}(x_k) + \frac{1}{2}h_k f_{t+1}^2(x_k)) + \Omega(f_{t+1}$
,其中$g_k = \partial_{\bar{y}_k} L(y_k, \bar{y}_k)$,$h_k = \partial_{\bar{y}_k}^2 L(y_k, \bar{y}_k)$

到这边已经差不多了,因为我们的模型是树模型,而树模型f_i是由多个叶子节点$w_1, w_2, \ldots, w_{T_i}$组成, 令 $I_j = \{i|q(x_i) = j\}$ 表示第i个样本落入到了第j个叶子节点中.则:

$min\ \Sigma_{k=1}^{n}(g_k f_{t+1}(x_k) + \frac{1}{2}h_k f_{t+1}^2(x_k)) + \Omega(f_{t+1})$

$\rightarrow min\Sigma_{j=1}^{T_{t+1}}[(\Sigma_{i\in I_j} g_i)w_j + \frac{1}{2}(\Sigma_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T_{t+1}$

$\rightarrow w_j^* = -\frac{\Sigma_{i\in I_j} g_i}{\Sigma_{i\in I_j} h_i + \lambda}$

对应的最优值就是: $-\frac{1}{2}\Sigma_{j=1}^{T_{t+1}}\frac{(\Sigma_{i\in I_j} g_i)^2}{\Sigma_{i\in I_j} h_i + \lambda} + \gamma T_{t+1}$

- **这个值被用来评估决策树的不纯洁性,类似于Gini指数等的作用,而这也确实是XGBoost的Splitting指标[13].**

如果分割过后,我们左右子树的值的和相比于原先的值有增加并且大于某一个阈值,那么我们就寻找能获取最大值的分割点进行分割,反之我们就不进行分割. 即:

$$L_{split} = \frac{1}{2}[\frac{(\Sigma_{i\in I_L} g_i)^2}{\Sigma_{i\in I_L} h_i+\lambda} + \frac{(\Sigma_{i\in I_R} g_i)^2}{\Sigma_{i\in I_R} h_i+\lambda} - \frac{(\Sigma_{i\in I} g_i)^2}{\Sigma_{i\in I} h_i+\lambda}] - \gamma$$

XGBoost初始算法(Exact Greedy Algorithm)

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i\in I} g_i$, $H \leftarrow \sum_{i\in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

版本 3 发布于2017-11-10 18:57