djjowfy

Blog address: http://djjowfy.com/2017/08/01/XGBoost%E7%9A%84%E5%8E%9F%E7%90%86/

XGBoost的原理

2017-08-01 | 原创 , 机器学习 | 4998

这篇博客的由来（瞎扯）

我在学习机器学习的时候，发现网上很少有对XGBoost原理探究的文章。而XGBoost用途是很广泛的。据kaggle在2015年的统计，在29只冠军队中，有17只用的是XGBoost，其中有8只只用了XGBoost。于是只能自己在网上找资料，幸而XGBoost的作者陈天奇在arixv上发布了一篇关于XGBoost的论文，于是就有了这篇博客。这篇博客首先将回顾监督学习，给出它的通用的优化函数。然后介绍回归树，它是XGBoost里的得到的最终模型的基本组成单元，许多棵回归树组成的回归森林就是XGBoost最终的学习模型。进而为了构造回归树，介绍了gradient tree boosting。从而引出了两种算法，一种是用于单线程的贪婪算法，一种是可以并行的近似算法，并作了结果的对比，显示出近似算法比较高的精确性。最后将介绍XGBoost的用法。

监督学习的回顾（背景知识）

概念

符号	含义
R^d	特征数目为d的数据集
$x_i \in R^d$	第i个样本
w_j	第j个特征的权重
$\hat{y}_i$	x_i的预测值
y_i	第i个训练集的对应的标签
Θ	特征权重的集合，$\Theta = \{w_j \vert j = 1, \cdots, d\}$

模型

基本上相关的所有模型都是在下面这个线性式子上发展起来的

$$\hat{y}_i = \sum_{j=0}^{d} w_j x_{ij}$$

上式中$x_0 = 1$，就是引入了一个偏差量，或者说加入了一个常数项。由该式子可以得到一些模型：

- **线性模型**，最后的得分就是$\hat{y}_i$
- **logistic模型**，最后的得分是$1/(1+exp(-\hat{y}_i))$。然后设置阀值，转为正负实例。
- **其余**的大部分也是基于$\hat{y}_i$做了一些运算得到最后的分数

参数

参数就是Θ，这也正是我们所需要通过训练得出的。

训练时的目标函数

训练时通用的目标函数如下：

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

在上式中$L(\Theta)$代表的是训练误差，表示该模型对于训练集的匹配程度。$\Omega(\Theta)$代表的是正则项，表明的是模型的复杂度。

训练误差可以用$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$来表示，一般有方差和logistic误差。

- 方差: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
- logstic误差: $l(y_i, \hat{y}_i) = y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i})$

正则项按照Andrew NG的话来说，就是避免过拟合的。为什么能起到这个作用呢？正是因为它反应的是模型复杂度。模型复杂度，也就是我们的假设的复杂度，按照奥卡姆剃刀的原则，假设越简单越好。所以我们需要这一项来控制。

- L2 范数: $\Omega(w) = \lambda ||w||^2$
- L1 范数(lasso): $\Omega(w) = \lambda ||w||_1$

常见的优化函数有有岭回归，logstic回归和Lasso，具体的式子如下

- 岭回归，这是最常见的一种，由线性模型，方差和L2范数构成。具体式子为$\sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda ||w||^2$
- logstic回归，这也是常见的一种，主要是用于二分类问题，比如爱还是不爱之类的。由线性模型，logistic 误差和L2范数构成。具体式子为$\sum_{i=1}^{n} [y_i ln(1 + e^{-w^T x_i}) + (1 - y_i) ln(1 + e^{w^T x_i})] + \lambda ||w||^2$

lasso比较少见，它是由线性模型，方差和L1范数构成的。具体式子为$\sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda ||w||_1$

我们的目标的就是让$Obj(\Theta)$最小。那么由上述分析可见，这时必须让$L(\Theta)$和$\Omega(\Theta)$都比较小。而我们训练模型的时候，根据Andrew Ng的课程，要在bias和variance中间找平衡点。bias由$L(\Theta)$控制，variance由$\Omega(\Theta)$控制。欠拟合，那么$L(\Theta)$和$\Omega(\Theta)$都会比较大，过拟合的话$\Omega(\Theta)$会比较大，因为模型的扩展性不强，或者说稳定性不好。

回归树的介绍（基础学习模型）

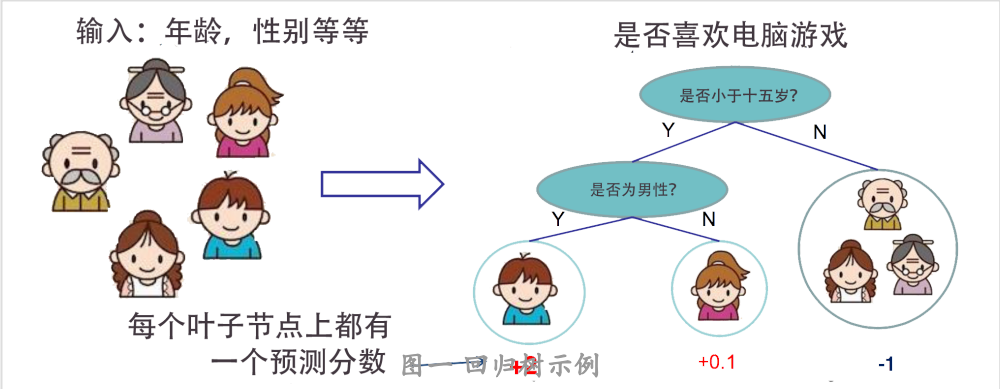
概述

回归树，也叫做分类与回归树，我认为就是一个叶子节点具有权重的二叉决策树。它具有以下两点特征

- 决策规则与决策树的一样

- 每个叶子节点上都包含了一个权重，也有人叫做分数

下图就是一个回归树的示例：



图一 回归树示例

回归树有以下四个优点：

1. 使用范围广，像GBM，随机森林等。(PS:据陈天奇大神的统计，至少有超过半数的竞赛优胜者的解决方案都是用回归树的变种)

2. 对于输入范围不敏感。所以并不需要对输入归一化

3. 能学习特征之间更高级别的相互关系

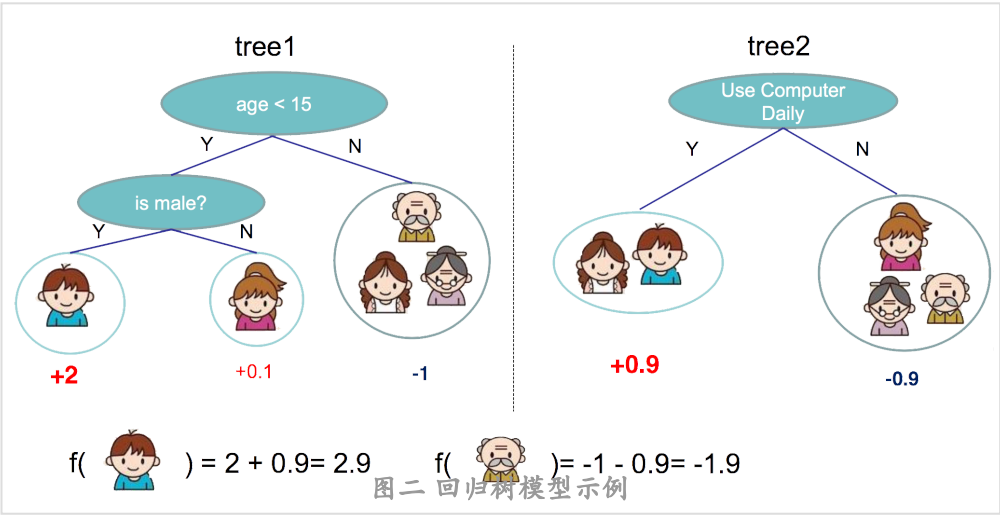
4. 很容易对其扩展

模型

假设我们有K棵树，那么

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

上式中$\mathcal{F}$表示的是回归森林中的所有函数空间。$f_k(x_i)$表示的就是第i个样本在第k棵树中落在的叶子的权重。以下图为例



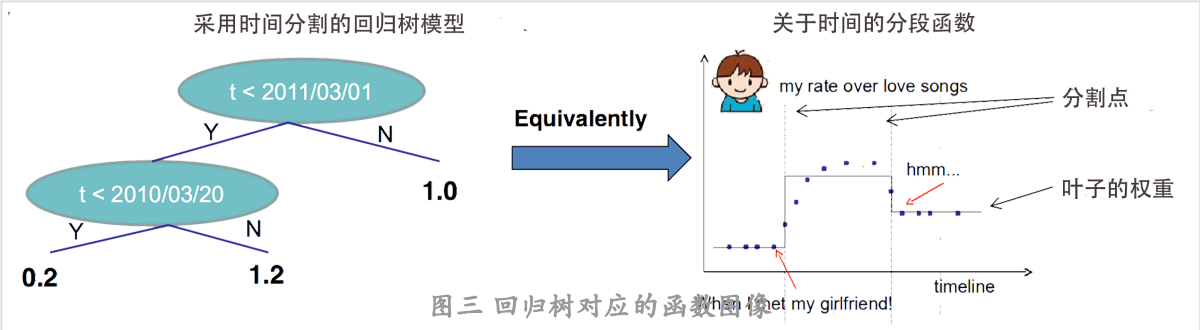
图二 回归树模型示例

可见小男孩落在第一棵树的最左叶子和第二棵树的最左叶子，所以它的得分就是这两片叶子的权重之和，其余也同理。

那么现在我们需要求的参数就是每棵树的结构和每片叶子的权重，或者简单的来说就是求f_k。那么为了和上一节所说的通用结构统一，可以设

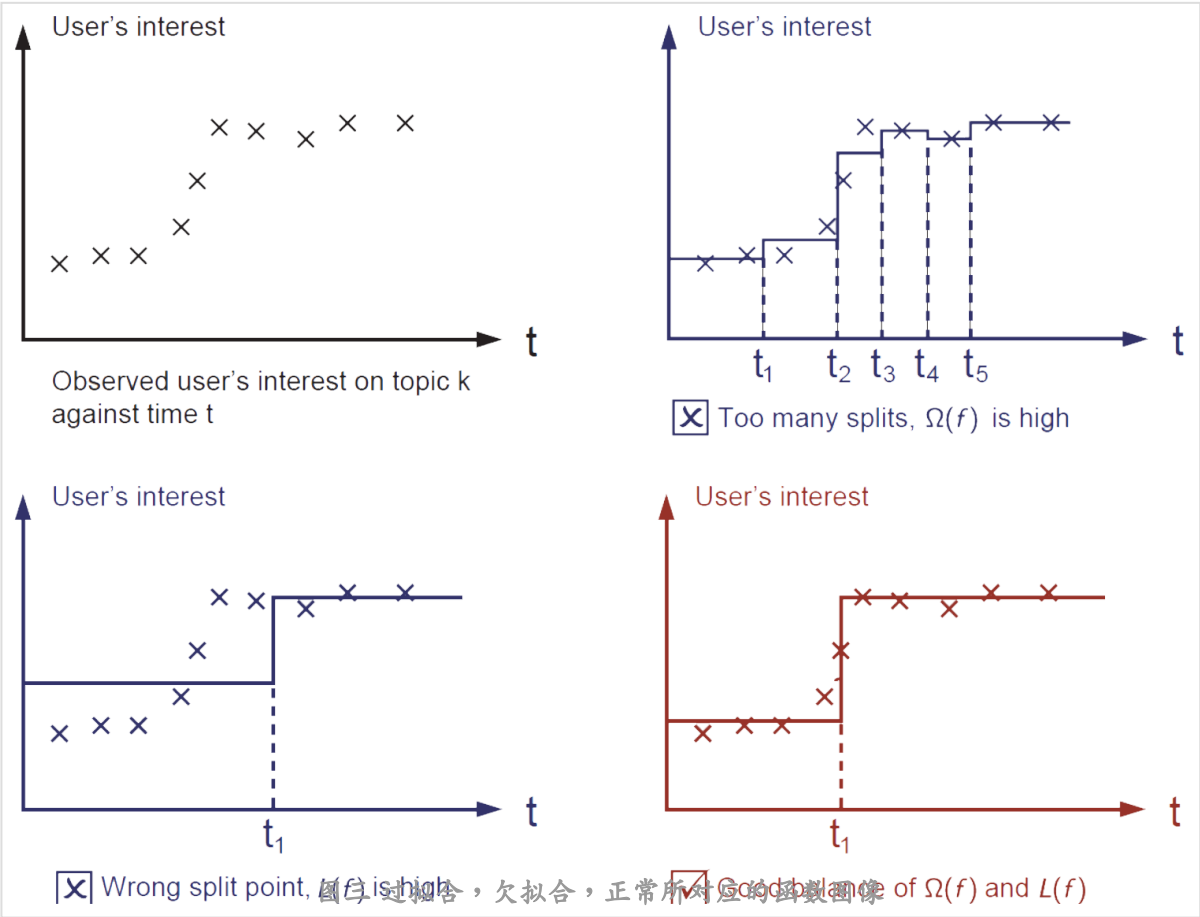
$$\Theta = \{f_1, f_2, f_3, \cdot, f_k\}$$

如果我们只看一棵回归树，那么它可以绘成分段函数如下



图三 回归树对应的函数图像

可见分段函数的分割点就是回归树的非叶子节点，分段函数每一段的高度就是回归树叶子的权重。那么就可以直观地看到欠拟合和过拟合曲线所对应的回归树的结构。根据我们上一节的讨论，$\Omega(f)$表示模型复杂度，那么在这里就对应着分段函数的琐碎程度。$L(f)$表示的就是函数曲线和训练集的匹配程度。



图四 过拟合，欠拟合，正常所对应的函数图像

综上所述，我们可以得出该模型的表达式如下

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

训练时的目标函数

训练误差如下

$$L(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) = \sum_{i=1}^{n} l(y_i, \sum_{k=1}^{K})$$

模型复杂度如下

$$\Omega(\Theta) = \sum_{k=1}^{K} \Omega(f_k)$$

因此，训练时的目标函数如下

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

如果训练误差

- $l(y, \hat{y}_i) = (y_i - \hat{y}_i)^2$ ，那么这就叫做gradient boosted machine

- $l(y, \hat{y}_i) = y_i ln(1 + e^{-\hat{y}_i} + (1 - y_i) ln(1 + e^{\hat{y}_i}))$ ，那么这就叫做logistBosst

对于$\Omega(f_k)$来说，可以用树的节点个数，树的深度，树叶权重的L2范数等等来进行描述。

参数

于是现在未知的就是f_k，这就是我们下一节所要解决的问题

$$\Theta = \{f_1, f_2, f_3, \cdots, f_k\}$$

Gradient Boosting(如何构造回归树)

上一节说明来回归树长啥样，也就是我们的模型最后长啥样。但是该模型应该怎么去求出Θ呢？这一节就介绍两种算法，一种是贪心算法，一种是近似算法。

贪心算法

完善目标函数的定义

这个算法的思想很简单，一棵树一棵树地往上加，一直到K棵树停止。过程可以用下式表达：

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

$\hat{y}_i^{(t)}$表示的是第i次循环后，对x_i所得到的得分。于是带入目标函数可得

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

可由泰勒公式得到下式

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$$

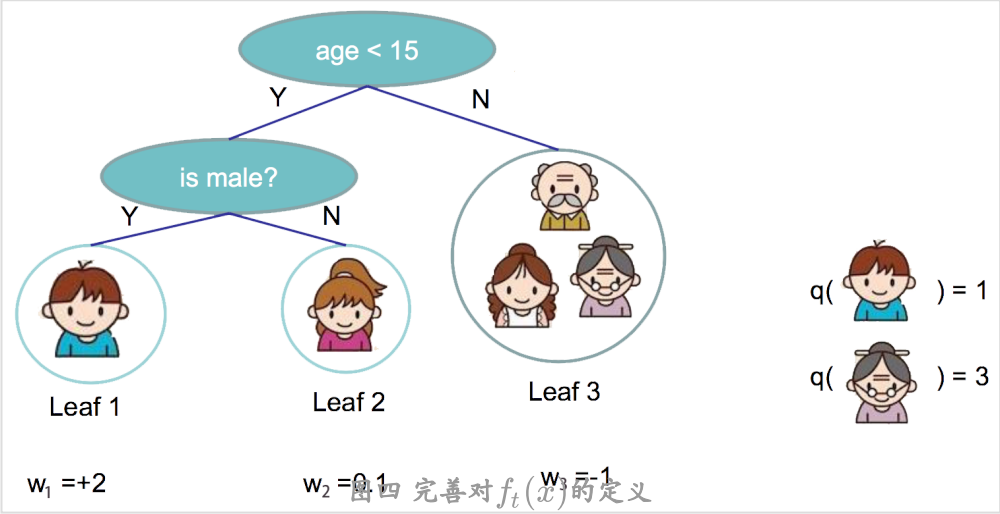
那么现在可以把$\hat{y}_i^{(t)}$看成上式中的$f(x + \Delta x)$，$\hat{y}_i^{(t-1)}$就是$f(x)$，$f_t(x_i)$为Δx。然后设g_i代表$f'(x)$，也就是$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$，用$h_i$代表$f''(x)$，于是$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$于是现在目标函数就为下式：

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant \\ &= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + [\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)}) + constant] \end{aligned}$$

很明显，上式中后面那项$[\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)}) + constant]$对于该目标函数我们求最优值点的时候并无影响，所以，现在可把优化函数写为

$$Obj^{(t)} \approx \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

上一节讨论了$f_t(x)$的物理意义，现在我们对其进行数学公式化。设$w \in R^T$，w为树叶的权重序列，$q : R^d \to \{1, 2, \cdots, T\}$，$q$为树的结构。那么$q(x)$表示的就是样本$x$所落在树叶的位置。可以用下图形象地表示



图四 完善对$f_t(x)$的定义

于是$f_t(x)$可以用下式进行表示

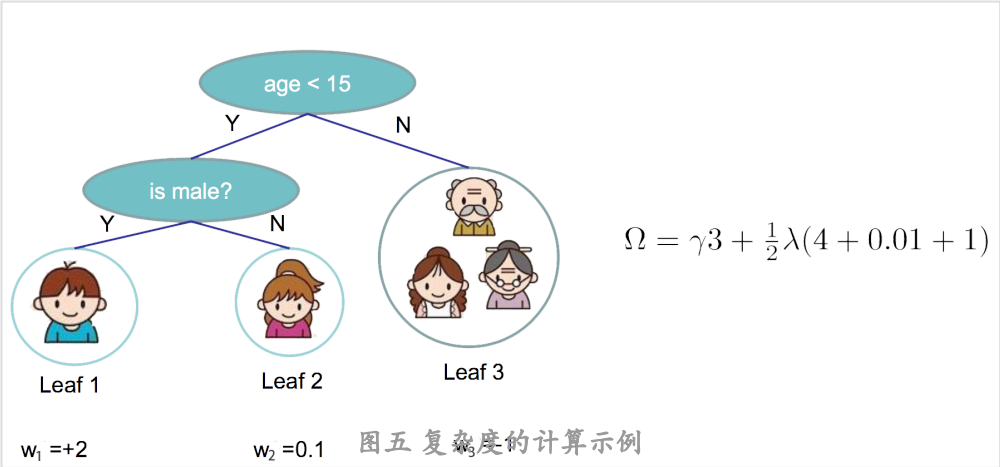
$$f_t(x) = w_{q(x)}, w \in R^T \; q : R^d \to \{1, 2, \cdots, T\}$$

现在对训练误差部分的定义已经完成。那么对模型的复杂度应该怎么定义呢？

树的深度？最小叶子权重？叶子个数？叶子权重的平滑程度？等等有许多选项都可以描述该模型的复杂度。为了方便，现在用叶子的个数和叶子权重的平滑程度来描述模型的复杂度。可以得到下式：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

上式中前一项用叶子的个数乘以一个收缩系数，后一项用L2范数来表示叶子权重的平滑程度。下图就是计算复杂度的一个示例：



图五 复杂度的计算示例

最后再增加一个定义，用I_j来表示第j个叶子里的样本集合。也就是图四中，每第i个圈，就用I_j来表示。

$$I_j = \{i|q(x_i) = j\}$$

好了，最后把优化函数重新按照每个叶子组合,并舍弃常数项：

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x)}^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \end{aligned}$$

求最优值

初中时所学的二次函数的最小值可以推广到矩阵函数里

$$min_x \;\; Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H} \quad , \quad H > 0$$

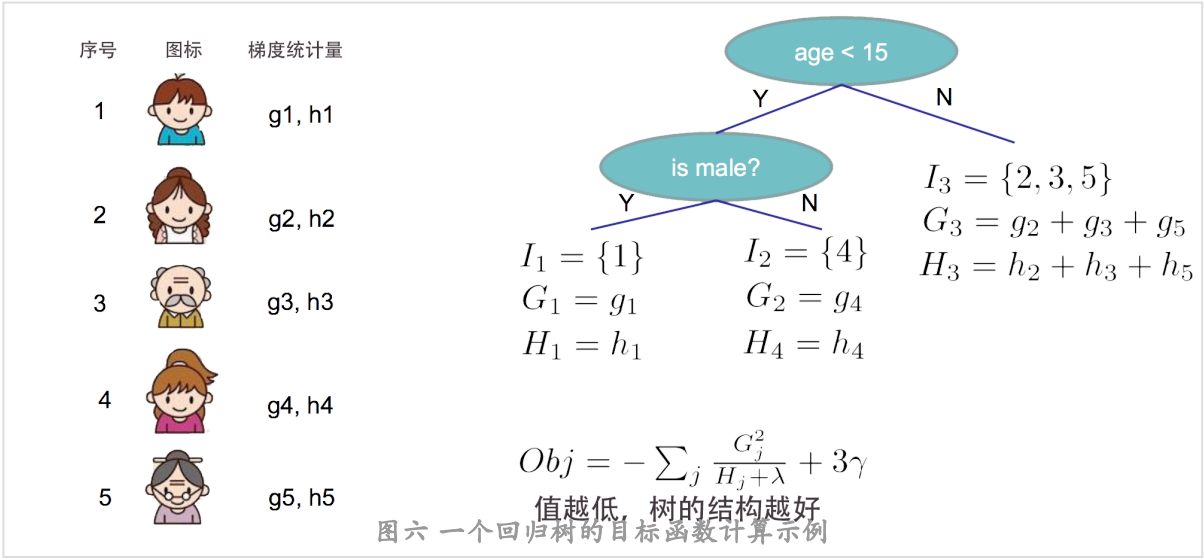
设$G_j = \sum_{i\in I_j} g_i$, $H_j = \sum_{i\in I_j} h_i$，那么

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \\ &= \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T \end{aligned}$$

因此，若假设我们的树的结构已经固定，就是$q(x)$已经固定，那么

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

为了形象地理解，下图就是一个示例：



图六 一个回归树的目标函数计算示例

求树结构

现在只要知道树的结构，就能得到一个该结构下的最好分数。可是树的结构应该怎么确定呢？没法用枚举，毕竟可能的状态基本属于无穷种。

这种情况，贪婪算法是个好方法。从树的深度为0开始，每一节点都遍历所有的特征。对于某个特征，先按照该特征里的值进行排序，然后线性扫描该特征来决定最好的分割点，最后在所有特征里选择分割后，$Gain$最高的那个特征。

$$\begin{aligned}
Obj_{split} &= -\frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda}] + \gamma T_{split} \\
Obj_{noSplit} &= -\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda} + \gamma T_{noSplit} \\
Gain &= Obj_{noSplit} - Obj_{split} \\
&= \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma(T_{split} - T_{nosplit})
\end{aligned}$$

这时，就有两种后续。一种是当最好的分割的情况下，$Gain$为负时就停止树的生长，这样的话效率会比较高也简单，但是这样就放弃了未来可能会有更好的情况。另外一种就是一直分割到最大深度，然后进行修剪，递归得把划分叶子得到的Gain为负的收回。一般来说，后一种要好一些，于是我们采用后一种，完整的算法如下（没有写修剪）

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

图七 贪婪算法

算法复杂度

1. 按照某特征里的值进行排序，复杂度是$O(nlog\,n)$
2. 扫描一遍该特征所有值得到最优分割点，因为该层（兄弟统一考虑）一共有n个样本，所以复杂度是$O(n)$
3. 一共有d个特征，所以对于一层的操作，复杂度是$O(d(nlog\,n+n)) = O(d\,nlog\,n)$
4. 该树的深度为k。所以总复杂度是$O(k\,d\,nlog\,n)$

注意事项

对某个节点的分割时，是需要按某特征的值排序，那么对于无序的类别变量，就需要进行one-hot化。不然的话，假设某特征有1，2，3三种变量。我们进行比较时，就会只比较左子树为1,2或者右子树为2,3，或者不分割，哪个更好。这样的话就没有考虑，左子树为1,3的分割。

因为$Gain$于特征的值范围是无关的，它采用的是已经生成的树的结构与权重来计算的。所以不需要对特征进行归一化处理。

回顾和完善

1. 每次循环增加一棵树
2. 在每次循环的开始时计算$g_i = \partial_{\hat{y}^{(t-1)}}\, l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial^2_{\hat{y}^{(t-1)}}\, l(y_i, \hat{y}^{(t-1)})$
3. 采用贪婪算法生长树$f_t(x)$,$Ohj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$
4. 把$f_t(x)$加在模型之中$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \epsilon f_t(x)$。注意，这里多了个$\epsilon$算子，这个是作为一个收缩系数，或者叫做步进。加上它的好处就是每一步我们都不是做一个完全的最优化，留下余地给未来的循环，这样能防止过拟合。

以上就是贪婪算法。显而易见，这种算法并行的效率很低，我们常用的scikit-learn等用的就是这个算法。XGBoost在单线程版本的时候，用的也是这种算法。

近似算法

根据前面的讨论，我们可以发现我们的模型对特征中的值的范围不敏感，只对顺序敏感。举个例子，假设一个样本集中某特征出现的值有1，4，6，7，那么把它对应的换成1, 2，3，4。生成的模型里树的结构是一样的，只不过对应的判断条件变了，比如把小于6换成了小于3而已。这也给我们一个启示，我们完全可以用百分比作为基础来构造模型。

我们用$\mathcal{D}_k = \{(x_{1k}, h_1), (x_{2k}, h_2), (x_{3k}, h_3), \cdots, (x_{nk}, h_n)$代表每个样本的第$k$个特征和其对应的二阶梯度所组成的集合。那么我们现在就能用百分比来定义下面的这个排名函数$r_k : \mathbb{R} \to [0, 1]$

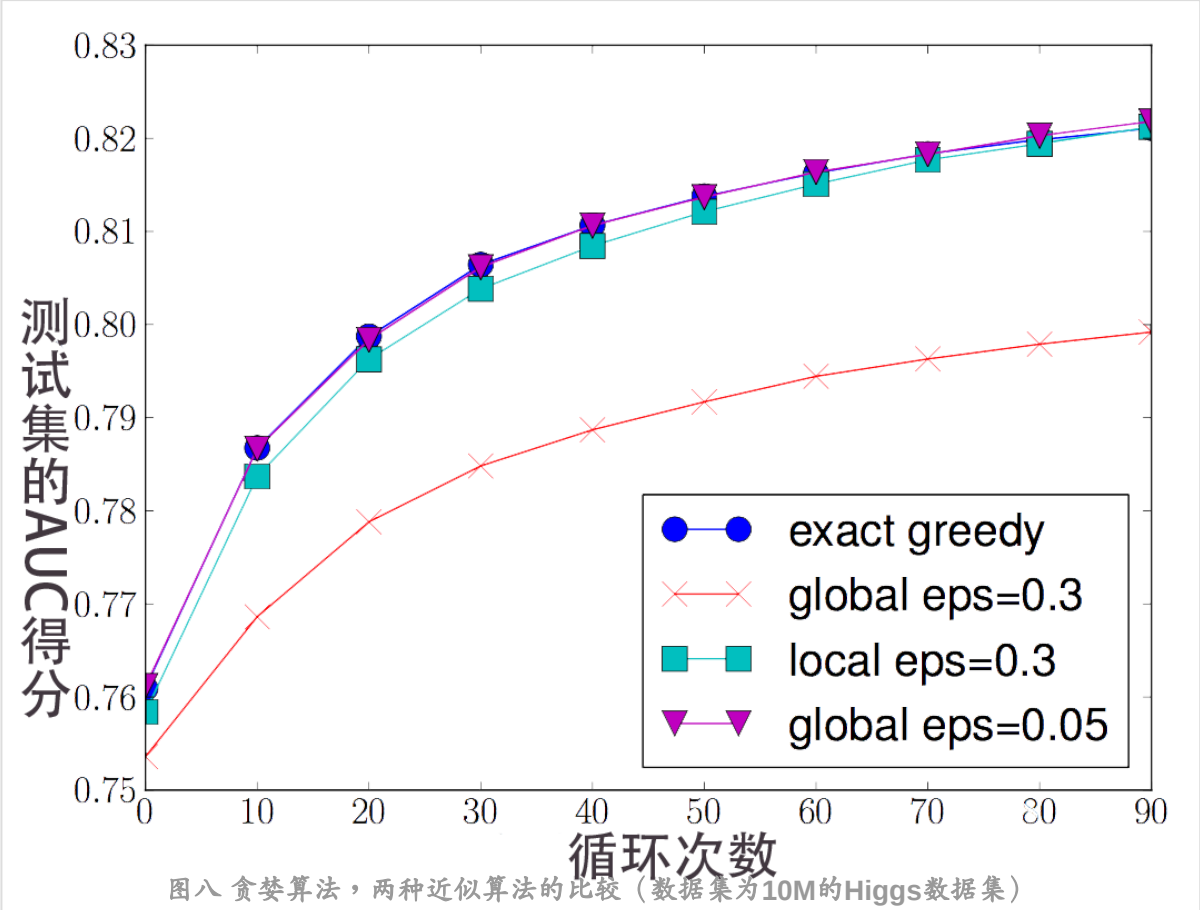
$$r_k(z) = \frac{1}{\sum_{(x,h)\in\mathcal{D}_k} h} \sum_{(x,h)\in\mathcal{D}_k, x<z} h$$

上式表示的就是该特征的值小于z的样本所占总样本的比例。于是我们就能用下面这个不等式来寻找分离点$\{s_{k1}, s_{k2}, s_{k3}, \cdot, \cdot, \cdot, s_{kl}\}$

$$\|r_k(s_{k,j}) - r_k(s_{k,j+1})\| < \epsilon, \min_i x_{ik}, \max_i x_{ik}$$

上式中ϵ表示的是一个近似比例，或者说一个扫描步进。就从最小值开始，每次增加$\epsilon * (\max_i x_{ik} - \min_i x_{ik})$作为分离点。然后在这些分离点中选择一个最大分数作为最后的分离点。

很明显$\mathcal{D}_k$有两种选择，或者说$\min_i x_{ik}$和$\max_i x_{ik}$有两种选择。一种是一开始选好，然后每次分离都不变，也就是说是在总体样本里选最大值和最小值。另外一种就是每次分离后，在分离出来的样本里选，也就是在以前的所定义的I_j里选。很容易就觉得后面这种选择方式虽然会繁琐一点，但是效果会比前面的那种好。现在我们定义前面的那种里的叫做全局选择，后面的这种叫做局部选择。陈天奇做了一个比较，曲线如下图：



图八 贪婪算法，两种近似算法的比较（数据集为10M的Higgs数据集）

由此可见，局部选择的近似算法的确比全局选择的近似算法优秀的多，所得出的结果和贪婪算法几乎不相上下。

算法的伪代码如下图所示

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

图九 近似算法的伪代码

一些进一步优化

在机器学习中，one-hot后，经常会得到的是稀疏矩阵，于是XGBoost也对这个作出了优化。还可以处理缺失值，毕竟这也是树模型一贯的优点。但这里就不细表了，毕竟太过于细节了。下一节我们就来看XGBoost这种强大的模型应该怎么使用吧。

使用

例程

官方例程如下:

```
1
2  import xgboost as xgb
3  # read in data
4  dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')
5  dtest = xgb.DMatrix('demo/data/agaricus.txt.test')
6  # specify parameters via map
7  param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }
8  num_round = 2
9  bst = xgb.train(param, dtrain, num_round)
10 # make prediction
11 preds = bst.predict(dtest)
```

参数

很明显，上面重要的就是param，这个参数应该怎么设。在官网上有整整一个网页的说明。在这里我们只挑选一些重要常用的说一下。

与过拟合有关的参数

在机器学习中，欠拟合很少见，但是过拟合却是一个很常见的东西。XGBoost与其有关的参数也不少。

增加随机性

- **eta** 这个就是学习步进，也就是上面中的ϵ。

- **subsample** 这个就是随机森林的方式，每次不是取出全部样本，而是有放回地取出部分样本。有人把这个称为行抽取，**subsample**就表示抽取比例

- **colsample_bytree**和**colsample_bylevel** 这个是模仿随机森林的方式，这是列抽取。**colsample_bytree**是每次准备构造一棵新树时，选取部分特征来构造，**colsample_bytree**就是抽取比例。**colsample_bylevel**表示的是每次分割节点时，抽取特征的比例。

- **max_delta_step** 这个是构造树时，允许得到$f_t(x)$的最大值。如果为0，表示无限制。也是为了后续构造树留出空间，和eta相似

控制模型复杂度

- **max_depth** 树的最大深度
- **min_child_weight** 如果一个节点的权重和小于这玩意，那就不分了
- **gamma**每次分开一个节点后，造成的最小下降的分数。类似于上面的Gain
- **alpha**和**lambda**就是目标函数里的表示模型复杂度中的L1范数和L2范数前面的系数

其他参数

- **booster** 表示用哪种模型，一共有gbtree, gbline, dart三种选择。一般用gbtree。
- **nthread** 并行线成数。如果不设置就是能采用的最大线程。
- **sketch_eps** 这个就是近似算法里的ϵ。
- **scale_pos_weight** 这个是针对二分类问题时，正负样例的数量差距过大。

参考文献

- XGBoost: A Scalable Tree Boosting System
- Introduction to Boosted Trees
- XGBoost官网

机器学习 # XGBoost # XGBoost的原理

‹ 对双调欧几里得旅行商问题的一些思考 对java中关于文件读取方法效率的比较 ›