xgboost

cloud

2016.1.8

1 前言

监督学习算法中，是由模型，参数，目标函数和优化算法这四个组件组成。

（1）模型：模型是给定样本 $x^{(i)}$ 来预测输出 $y^{(i)}$，比较常见的如线性模型 $y = h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$。在这里预测的 y 有不同的解释，可以作为回归目标的输出，或者进行 $sigmod$ 变换得到概率，或者作为排序的指标等。

（2）参数：参数就是我们要求解的东西，上面的 θ 系数就是我们要求解的参数。

（3）目标函数：要想求得较好的模型，就要去寻找一个比较好的参数，这个时候就引出了目标函数。一般的目标函数可以写成 $Obj(\Theta) = L(\Theta) + \Omega(\Theta)$，第一项表示的是损失函数，第二项表示正则化项。损失函数有平方损失，对数损失等，误差可以分解为偏差和方差，偏差期望的是所有样本预测值和真实值的差尽量小，方差期望的训练样本变动对预测结果的影响尽量小。正则化项就是我们常说的 L_1 和 L_2 正则，正则化项鼓励更为简单的模型，简单的模型不容易过拟合且预测更加稳定。

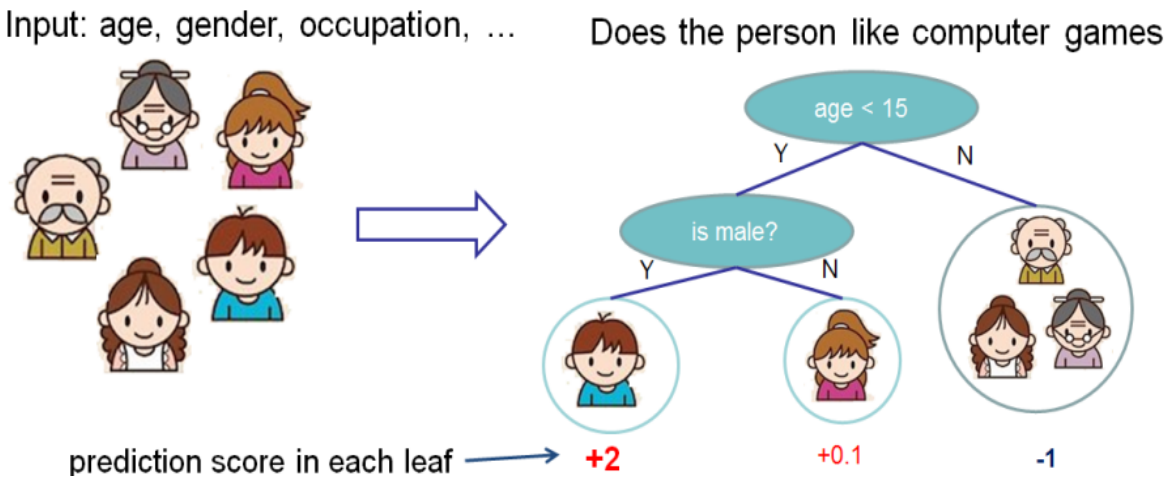
（4）优化算法：那么如何学习目标函数呢？有极大似然估计，贝叶斯估计，梯度下降，EM 算法等。但是很多时候我们往往只知道优化算法，而没有仔细考虑目标函数设计的问题，那么 xgboost 就是通过改进 Boosted Tree 的目标函数来加速求解参数的一种改进算法。

PS: 无论是 GBDT，GBRT 还是 MART 它们只是 Boosted Tree 不同形式的称呼而已，本质来说是一致的，所以这里统一称呼就是 Boosted Tree。

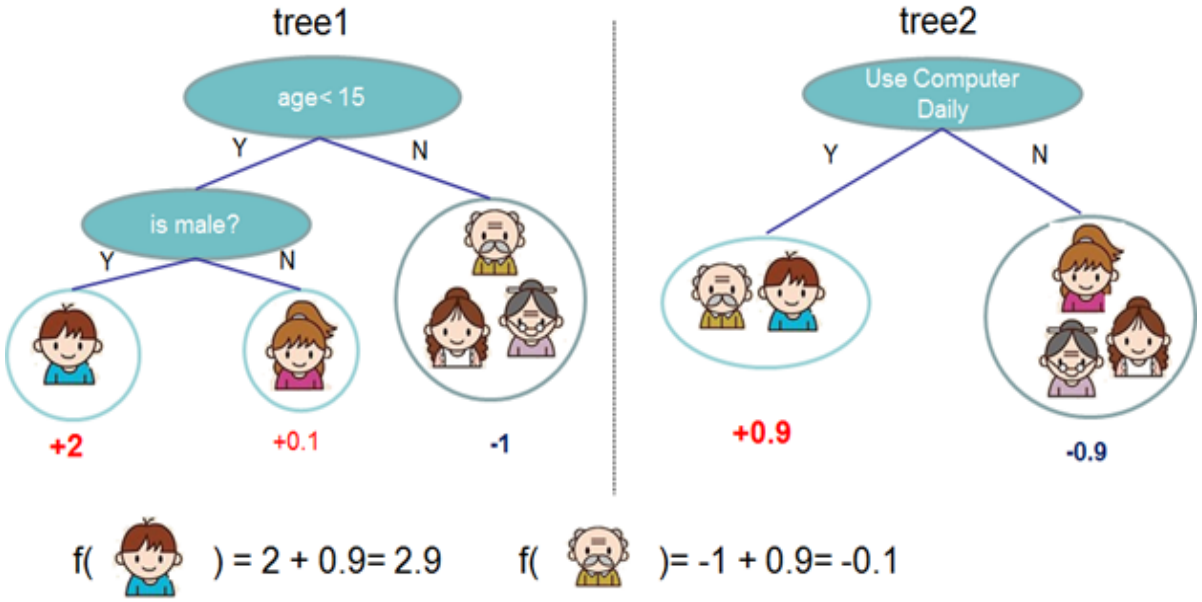
2 Boosted Tree

2.1 目标函数

Boosted Tree 最基本的组成部分是回归树也叫作 CART(classfication and regression tree)。



上面就是一个 CART 的例子，CART 会根据输入样本的属性分配到各个叶子节点，而每个叶子节点都会对应一个实数分数。在大多数情况下，一棵树由于过于简单无法有效的预测，所以就出现了整合方法 (Ensemble)，就是把多棵树预测的结果按照一定的权值加合起来，如下图所示。



GBDT 和 RF 都是基于 Tree Emsemble，只是学习模型参数的方法不一样。在 Tree Ensemble 中，参数对应树的结构，以及每个叶子节点上的预测分数。在这里不要把 Boosting，Bagging 和 Ensemble 弄晕了，Boosting 和 Bagging 方法是利用了 Ensemble 的思想。

对于 Tree Ensemble，数学模型如下，这里 $\mathcal{F}$ 对应了所有回归树的集合。

$$\hat{y}^{(i)} = \sum_{k=1}^{K} f_k(x^{(i)}) \quad f_k \in \mathcal{F}$$

进一步设计的目标函数如下，第一项是训练误差，第二项是每棵树的复杂度之和。

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

2.2 模型学习

上节中设计的目标函数的参数认为是在一个函数空间里，所以不能采用传统的随机梯度下降之类的算法来学习模型。所以我们采用前向分步算法，即每一次保留原来的模型不变，每次加入一个新的优化函数到模型中去，可以写成如下形式：

$$
\begin{aligned}
&\hat{y}_0^{(i)} = 0 \\
&\hat{y}_1^{(i)} = f_1(x^{(i)}) = \hat{y}_0^{(i)} + f_1(x^{(i)}) \\
&\hat{y}_2^{(i)} = f_1(x^{(i)}) + f_2(x^{(i)}) = \hat{y}_1^{(i)} + f_2(x^{(i)}) \\
&... \\
&\hat{y}_t^{(i)} = \sum_{k=1}^{t} f_k(x^{(i)}) = \hat{y}_{t-1}^{(i)} + f_t(x^{(i)})
\end{aligned}
$$

上式中最后一项 $\hat{y}_t^{(i)}$ 是第 t 轮的模型预测，$\hat{y}_{t-1}^{(i)}$ 是前 $t-1$ 轮的模型预测，$f_t(x^{(i)})$ 是在 t 轮新加入的函数。问题进一步变成如何选择每一轮加入的函数，显而易见每一轮选取的 f 要使目标函数尽可能的降低。那么第 t 轮的目标函数可以写成如下的形式，这里的损失函数考虑平方损失。

$$
\begin{aligned}
Obj(\Theta)^{(t)} &= \sum_{i=1}^{N} l(y^{(i)}, \hat{y}_t^{(i)}) + \sum_{k=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{N} l(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(x^{(i)})) + \Omega(f_t) + constant \\
&= \sum_{i=1}^{N} (y^{(i)} - (\hat{y}_{t-1}^{(i)} + f_t(x^{(i)}))^2 + \Omega(f_t) + constant \\
&= \sum_{i=1}^{N} (r - f_t(x^{(i)}))^2 + \Omega(f_t) + constant
\end{aligned}
$$

上式的 $r = y^{(i)} - \hat{y}_{t-1}^{(i)}$ 就是常说的残差。如果不是平方损失的情况，可以将一阶导数近似作为残差进行求解，GBDT 就是利用残差来求解模型的。

3 xgboost

3.1 目标函数

在 xgboost 中，我们采用二阶的泰勒展开式定义一个近似的目标函数，方便进行这一步的计算。泰勒展开式如下所示：

$$
\begin{aligned}
Obj(\Theta)^{(t)} &= \sum_{i=1}^{N} l(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t(x^{(i)})) + \Omega(f_t) + constant \\
&\simeq \sum_{i=1}^{N} [l(y^{(i)}, \hat{y}_{t-1}^{(i)}) + g_i f_t(x^{(i)}) + \frac{1}{2} h_i f_t^2(x^{(i)})] + \Omega(f_t) + constant
\end{aligned}
$$

上式中是利用泰勒展开 $f(x+\Delta x) \simeq f(x)+f'(x)\Delta x+\frac{1}{2}f''(x)\Delta x^2$，其中 $g_i = \frac{\partial l(y^{(i)}, \hat{y}_{t-1}^{(i)})}{\partial \hat{y}_{t-1}^{(i)}}$，$h_i = \frac{\partial^2 l(y^{(i)}, \hat{y}_{t-1}^{(i)})}{\partial (\hat{y}_{t-1}^{(i)})^2}$。如果把目标函数的常数项去掉，就会得到比较统一的目标函数：

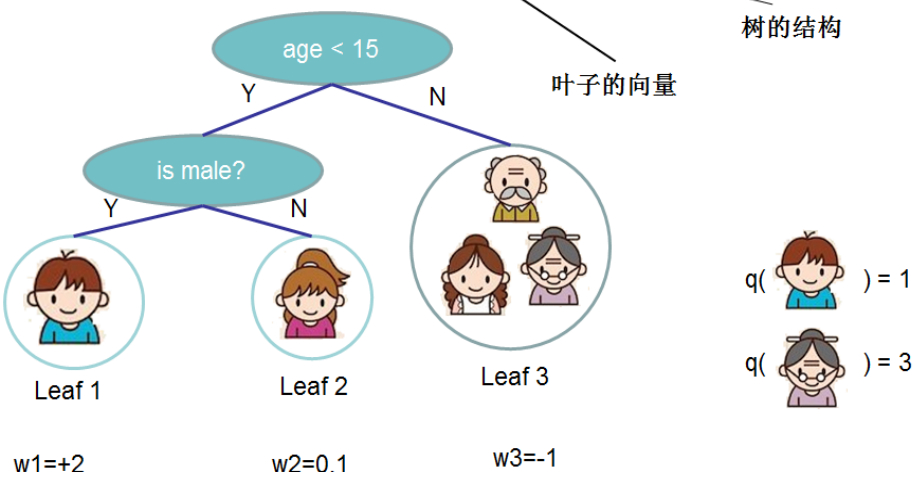
$$
Obj(\Theta)^{(t)} \simeq \sum_{i=1}^{N} [g_i f_t(x^{(i)}) + \frac{1}{2} h_i f_t^2(x^{(i)})] + \Omega(f_t)
$$

由上式可知它只依赖于每个数据点在损失函数的一阶和二阶导数。

3.2 树的复杂度

前面一直讨论的是目标函数的训练误差部分，接下来讨论如何定义树的复杂度即 $\Omega(f_t)$。把树拆分成结构部分 q 和叶子权重部分 w，下图就是一个具体的例子，结构函数 q 把输入 x 映射到叶子的索引号上面去，而 w 给定了每个索引号对应的叶子分数是什么。

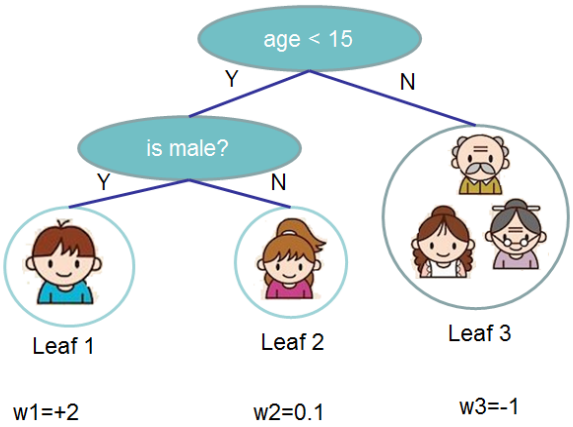
$$
f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}
$$



比如说小男孩代表一个样本 x，代入结构函数 $q(x) = 1$，那么 $f_t(x) = w_1 = +2$。进一步要定义树的复杂度如下图所示，其中 γ 和 λ 分别是叶子节点数 T 和正则化项的系数。

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

叶子的个数　　w的L2模平方



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

3.3 最终定义

在上节树的复杂度新的定义下，把目标函数进行改写如下，其中 I 被定义为每个叶子上面的样本集合 $I_j = \{i|q(x^{(i)}) = j\}$。

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

上式中 $G_j = \sum_{i\in I_j} g_i$，$H_j = \sum_{i\in I_j} h_i$。假设已经知道树的结构 q，可以通过这个目标函数求解出最优的 w 为：

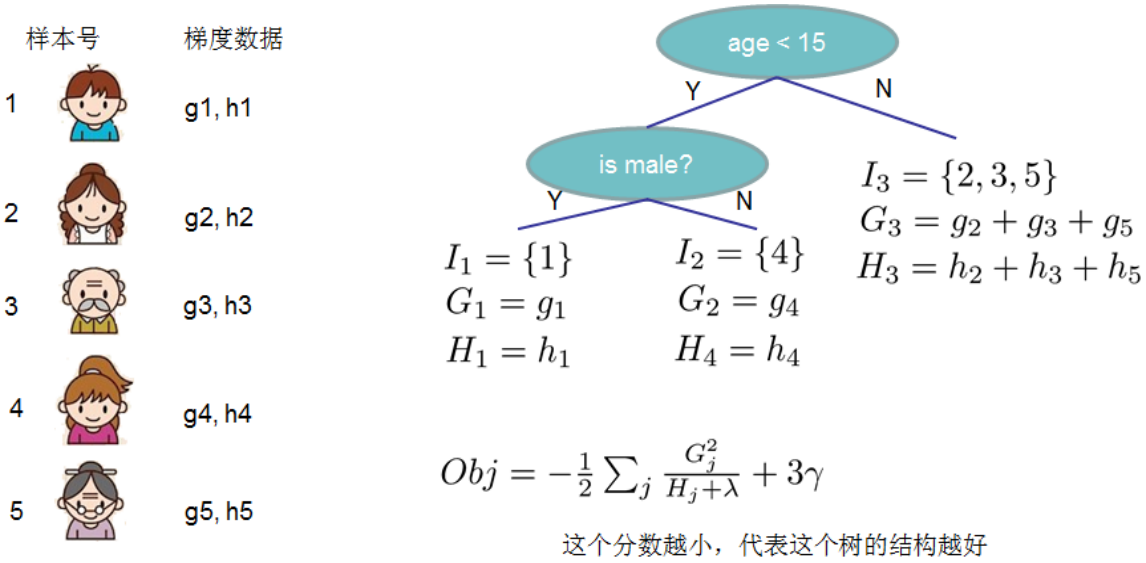
$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

那么最优的目标函数为：

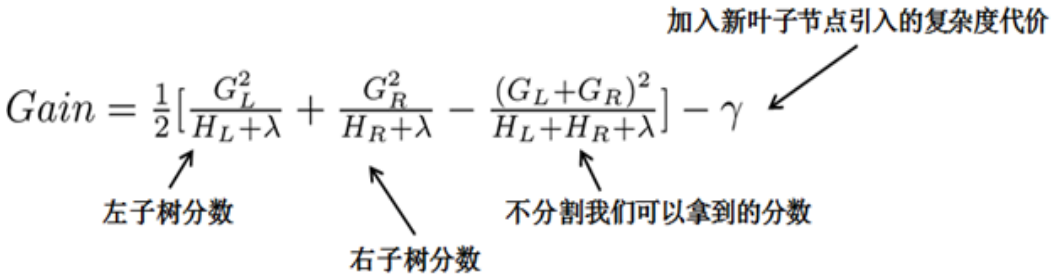
$$Obj(\Theta) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

3.4 示例

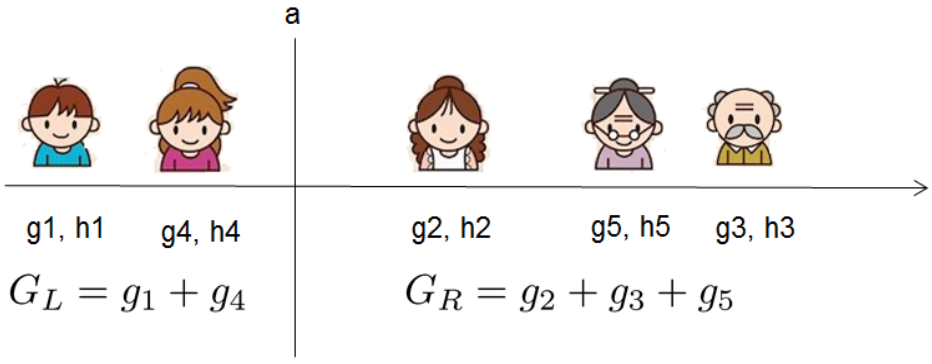
目标函数代表了当我们指定一个树的结构时，一种类似 Gini 系数特定对于树结构进行打分的函数，下图是一个具体的打分函数计算的例子。



所以 xgboost 也很简单，就是不断枚举不同树的结构，利用这个打分函数来寻找一个最优结构的树加入到模型中去。一般采用贪心策略，每一次尝试对已有的叶子加入一个分割，具体的评价公式如下：



而在进行分割的时候也有枚举所有可能的分割方案，如何高效地枚举所有的分割呢？假设要枚举所有 $x < a$ 这样的条件，对于某个特定的分割 a 要计算 a 左边的导数和以及右边的导数和，如下所示：



对于所有的 a，只需做一遍从左到右的扫描就可以枚举出所有分割的梯度 G_L 和 G_R，然后利用上面的公式计算每个分割方案的分数即可。

观察这个目标函数，大家会发现第二个值得注意的事情就是引入分割不一定会使得情况变好，因为我们有一个新叶子的惩罚项。优化这个目标对应了树的剪枝，当引入的分割带来的增益小于一个阀值的时候，我们可以剪掉这个分割。大家可以发现，当我们正式地推导目标的时候，像计算分数和剪枝这样的策略都会自然地出现。

4 GBDT 和 xgboost 对比

（1）传统 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，这个时候 xgboost 相当于带 L_1 和 L_2 正则化项的逻辑回归（分类问题）或者线性回归（回归问题）。

（2）传统 GBDT 在优化时只用到一阶导数信息，xgboost 则对损失函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。

（3）xgboost 在损失函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的 $score$ 模的平方和。正则项降低了模型的方差，使学习出来的模型更加简单，防止过拟合，这也是 xgboost 优于传统 GBDT 的一个特性。

（4）$Shrinkage$（缩减）相当于学习率。xgboost 在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把学习率设置得小一点，然后迭代次数设置得大一点。同时传统 GBDT 的实现也有学习率。

（5）列抽样。xgboost 借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是 xgboost 异于传统 gbdt 的一个特性。

（6）对缺失值的处理。对于特征的值有缺失的样本，xgboost 可以自动学习出它的分裂方向。

（7）xgboost 工具支持并行。boosting 是一种串行的结构，那么 xgboost 怎么并行的？xgboost 也是一次迭代完才能进行下一次迭代的。xgboost 的并行是在特征粒度上的，决策树的学习最耗时的一个步骤就是对特征的增益值进行排序（因为要确定最佳分割点），xgboost 在训练之前预先对数据进行了排序，然后保存为 $block$ 结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个 $block$ 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

（8）可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以 xgboost 还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。

5 参考文献

1.http://www.52cs.org/?p=429

2.https://www.zhihu.com/question/41354392